# The DOLFIN Project

An application report on a consistent design and verification flow for a large digital neural network.

A.Wassatsch, M.Haase, D. Timmermann
University of Rostock, Dept. EE and IT
Institute of Applied Microelectronics and CS
R.-Wagner-Str. 31, D-18119 Rostock, Germany
wassatsch@e-technik.uni-rostock.de

February 9, 2000

*Keywords*— **VHDL, VSS, Cyclone, large design verification, hierarchical synthesis, rapid prototyping**

## Abstract

In this application report we describe the development process of a new concept for implementing artificial neural networks. Based on the idea to utilize a previously developed concept of digit online calculation of arithmetic functions we start the development with high level simulations of Matlab network models. We also use these simulation models to verify the effects of modifications in the implementation of the network. At this point we start the VHDL network implementation based on digit online architecture library elements. Assisted by the results of the soft model simulation we adjust the VHDL implementation. Unfortunately, only VHDL models for very small network examples could be successfully validated by VSS and Cyclone due to the complexity of the description. Therefore, as the next step of design verification we have developed an emulation environment for an Aptix MP3C rapid prototyping system.

In the presentation we will demonstrate our designflow and solutions for the concatenation of the different software tools used. Using a full backpropagation network as an example we will discuss our results. We will also present our mechanism to automate the preparation and realization for a successful verification.

## 1 Introduction

Artificial neural networks are an excellent example for massive parallel data processing. The implementation of fully parallel data processing algorithms is mainly ressource limited. Therefore, in most cases only a small inner loop of the whole data processing unit is implemented. This inner loop consists of arithmetic operations like multiplications and additions. To execute the whole algorithm the operations in the inner loop must be sequentially processed.

Alternatively, the use of MSD (most significant digit) first serial arithmetic is a promising way to speedup the overall operation. In this paper we describe the difficulties which have to be solved during the evaluation of the novel architecture.

First, we give a short outline of the arithmetic algorithms in section 2. In section 3 we refurbish the basic principles of neural networks. In section 4 the concept of our network is described. The first verification model will be discussed in subsection 4.2. In subsection 4.3 we describe the problems which led to the development of our solution which is presented in subsection 4.5.

We generalize the problem and our solution in section 5. The section 6 is dedicated to the conclusion.

# 2 Digit Online Arithmetic

Before we can start with introducing the basics of digit online we need at first some knowledge about an alternative number representation system. This redundant number system is required to fulfill the necessary operation result updates later in the algorithm.

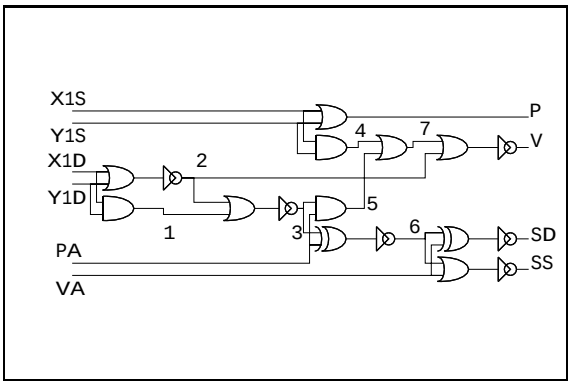


**Figure 1:** SD Takagi adder

## 2.1 Redundant Arithmetic

The term "redundant arithmetic" characterizes the utilization of a redundant number system for the number representation. It does not mean that the arithmetic circuits are robust against internal failure.

Like nonredundant number systems the base value $r$ of the number representation can be freely chosen. Given a number base $r = 2$ the commonly utilized redundant binary number system allows the digit values $\bar{1}, 0, 1$.

$$X = \sum_{i=-n}^{m} X_i = \sum_{i=-n}^{m} x_i * r^i \quad (1)$$

$$x_i \in \{\overline{(r-1)}, ..0, ..(r-1)\} \quad (2)$$

Due to the redundant number representation one value can have many different signed digit (SD) vector equivalencies. For instance the value 3 can have the following representations in a 4-digit vector $0011, 010\bar{1}, 01\bar{1}1, 1\bar{1}0\bar{1}, 1\bar{1}\bar{1}1$. This may pose some problems if you have to detect a special value, for instance in a comparator. There you first have to transform the redundant number representation into a nonredundant system, which simply can be done by a small and slow ordinary carry propagate adder (CPA).

A redundant adder (RA) is the basic module for the digit online algorithms. The main advantage of a redundant adder is a constant delay independent from word/vector length. On the other hand redundant adders like the Takagi adder [KET87] shown in Fig.1 for SD operands are slightly larger than nonredundant adders.

For our digit online implementation we prefer a SD representation due to the higher independence of the digit value from the total value. Indeed, carry save (CS) representations are feasible as well.

## 2.2 Digit Online Algorithm

Unlike most other serial arithmetic implementations digit online algorithms generate results starting with the most significant digit (MSD) first. Simple arithmetic functions like addition or multiplication can be implemented without any special adders in least significant digit (LSD) first manner. This is due to the resemblance of their hardware implementation to the corresponding paper-and-pencil algorithm. A MSD-first implementation is also possible when using redundant adders. Higher level arithmetic operations like division or square root are MSD-first algorithms by nature and can not be transformed to LSD-first without an extreme performance penalty. Due to the result generation starting with the most-significant-digit we can easily integrate this algorithm in a MSD-first organized stream processing architecture.

The idea of digit online algorithms, first presented in [ET77], can be described for a two

operand example with equation 3 by equation 4 and the following iteration equation 5

$$S = f(X,Y) \text{ with } X = \sum_{i=0}^{j+\delta} x_i r^{-i};\ Y = \sum_{i=0}^{j+\delta} y_i r^{-i} \quad (3)$$
$$\{x_i, y_i, s_i\} \in \{\bar{p}, \ldots, \bar{1}, 0, 1, \ldots p\}$$
$$\text{and } \frac{r}{2} \leq p \leq r-1$$

Initialization:

$$W_{(-\delta+1\ \ldots\ -1)} \leftarrow C \quad X_0 \leftarrow 0 \quad (4)$$
$$s_{(-\delta\ \ldots\ -1)} \leftarrow 0 \quad Y_0 \leftarrow 0$$

Recursion:

$$\begin{aligned} \text{for} \quad & j = 0, 1, \ldots, m \\ W_j \quad \leftarrow \quad & (W_{j-1} - s_{j-1}) + \\ & f(r^{-\delta}, x_j, X_{j-1}, y_j, Y_{j-1}) \quad (5) \\ s_{j-1} \quad \leftarrow \quad & S(W_j) \end{aligned}$$

and the selection function:

$$S(W_j) = \begin{cases} p, & if \quad w_s < W_j \\ 0, & if \quad -w_s \leq W_j \leq w_s \\ \bar{p}, & if \quad W_j < w_s \end{cases} \quad (6)$$

The algorithm starts with an initialization of some variables, the scaled residual $W_j$, the operand vectors $X_j$ and $Y_j$, and the first $\delta$-result digits $s$. Depending on the function the algorithm needs $\delta$ leading operand-digits to calculate the first result digit. This incorporates a function specific online delay $\delta$ for the result generation. During the following recursion the actual residual $W_j$ will be calculated and each iteration outputs a new result digit $s_j$ determined by the selection function (equation 6).

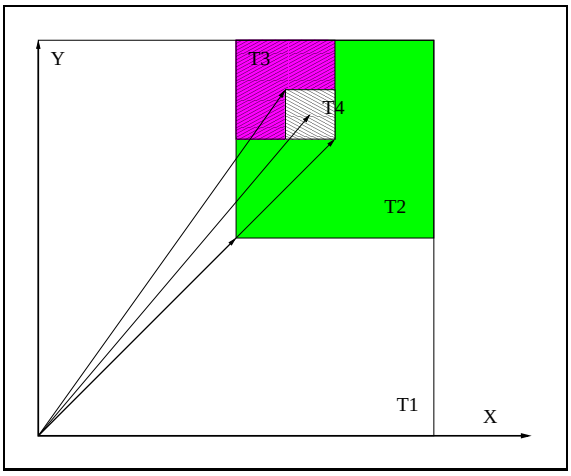


**Figure 2:** accuracy process for digit online linear function

The resulting digit vector approximates the result step-by-step as shown in Fig.2. The redundant number system allows for any necessary corrections of a previously estimated result.

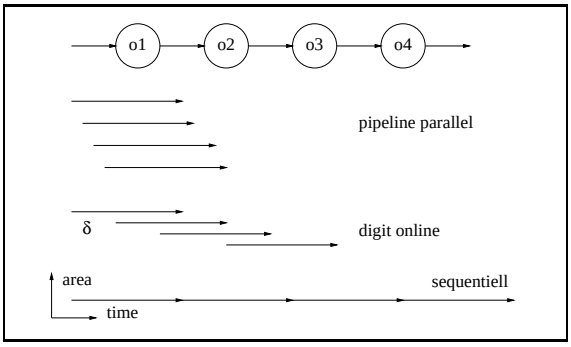


**Figure 3:** Cascading of digit online modules

By cascading digit online modules as shown in Fig.3 sequential operations can be massively overlapped [Erc84]. Compared to conventional implementations of successively LSD-first and MSD-first operations this results in dramatic performance improvements.

## 2.3 Basic Online Algorithm

The first basic algorithm for digit online calculation is the summation of two operands. We have two ways to develop an algorithm and an architecture for this operation. At first we can use the architecture of a parallel redundant adder to pipeline the calculation. Then we can reduce our focus to one pipeline stage and reorganize the communication between the stages to get an ar-

chitecture for a digit-wise calculation. The resulting structure is much simpler and smaller than the second version which we can develop by using the approach given in [Erc84]. This can be described by a recursion through the equations 7 ... 10.

$$S = f(X, Y) = X + Y \tag{7}$$

Initialization:

$$\begin{aligned} W_{-1} &\leftarrow 0 \\ s_{(-2)} = s_{(-1)} &\leftarrow 0 \end{aligned} \tag{8}$$

Recursion:

$$\begin{aligned} \text{for} \quad & j = 0, 1, \ldots, m+1 \\ W_j \quad \leftarrow \quad & (W_{j-1} - s_{j-2}) + r^{-\delta}(x_j + y_j) \\ s_{j-1} \quad \leftarrow \quad & S(W_j) \end{aligned} \tag{9}$$

and the selection function:

$$S(W_j) = \begin{cases} sign(W_j)\lfloor |W_j| + \frac{1}{2} \rfloor & if \quad |W_j \leq p| \\ sign(W_j)|W_j| & else \end{cases} \tag{10}$$

A second basic digit online algorithm calculates the multiplication of two operands. For multiplication the best way to develop a calculation scheme is the approach mentioned above, too. This results in a similar equation set (11 .. 14).

$$S = f(X, Y) = X * Y \tag{11}$$

Initialization:

$$\begin{aligned} W_{(-\delta+1 \ldots -1)} \leftarrow C \quad & X_0 \leftarrow 0 \\ s_{(-\delta \ldots -1)} \leftarrow 0 \quad & Y_0 \leftarrow 0 \end{aligned} \tag{12}$$

Recursion:

$$\begin{aligned} \text{for} \quad & j = 0, 1, \ldots, m \\ W_j \quad \leftarrow \quad & (W_{j-1} - s_{j-1}) + \\ & r^{-\delta}(Y_j * x_j + X_{j-1} * y_j) \\ s_{j-1} \quad \leftarrow \quad & S(W_j) \end{aligned} \tag{13}$$

and the selection function:

$$S(W_j) = \begin{cases} sign(W_j)\lfloor |W_j| + \frac{1}{2} \rfloor & if \quad |W_j \leq p| \\ sign(W_j)|W_j| & else \end{cases} \tag{14}$$

From these two basic online algorithms we can easily derive further operations like linear or square functions.

### 2.4 Advanced Online Algorithm

More advanced online functions are the online division and the online square root. Both operations are reduced to simple add, multiply, and shift operations. This is also the principle chosen to implement the digit online cordic function in [Sma92].

For further functions we can choose two different ways. At first we can use the derivation schema mentioned above to develop a new special digital online algorithm for the function we are looking for. Alternatively and faster we can reuse the already developed functions and modules to build an approximation of our function for instance by series expansion.

To get an impression about the different possible digit online functions in Tab.1 the characteristics for some modules are gathered.

| module name | clb slices | func. gen. | flip flops | io pins | gate equ. |
|---|---|---|---|---|---|
| add | 3 | 6 | 3 | 7 | 60 |
| mul8 | 321 | 642 | 136 | 11 | 5057 |
| exp8 | 307 | 613 | 60 | 5 | 4539 |
| outFunc | 82 | 163 | 76 | 5 | 1964 |

Table 1: Syntheses results for online components

## 3 Artificial Neural Network

Due to the numerous literature available on principles and implementations of neural networks like [RR91], [FS91], and [Zel97] we outline only those topics which are relevant for our implementation.

The operation on a neural network can be divided in general into two phases, the training or learning task and the evaluation task. A learning phase is necessary because the fixed storage of weights for evaluation of the synaptic connections is unusual and severely limits the adaptation of the network to changing conditions. Commonly, external memory served for storing some sort of learning patterns. Thus a different application requires only a new contents of the pattern memory.

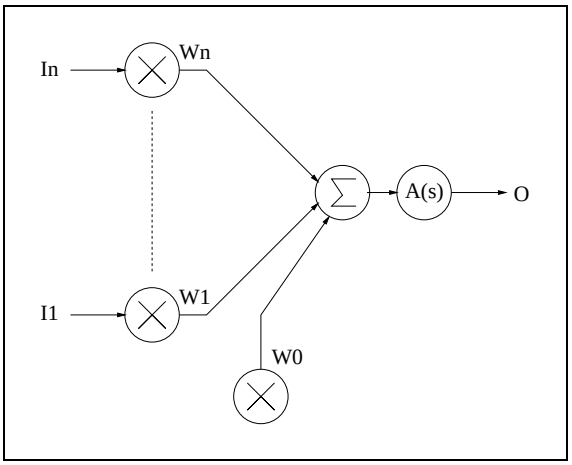


**Figure 4:** Principle of a neuron

The principle data flow of a simple neuron is shown in Fig.4. A neuron consist of $n$ stimulating and an additional inhibiting synapses, an accumulation and a evaluation unit. Each of this synapses has its own weight which represent the influence of this input signal on the neuron. In an implementation the weighting is commonly modeled by an ordinary multiplier. The activity of the neuron can be calculated by different algorithms like the sigmoid or the $\tanh$ function.

## 3.1 Structure

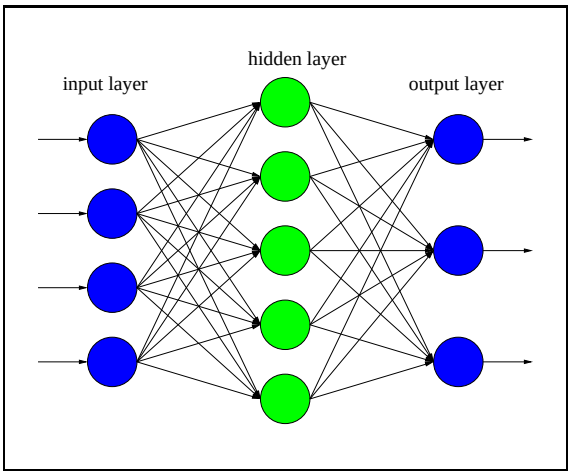


**Figure 5:** Principle of a neural feed-forward network

Depending on the network type multiple instances of that now can be connected together to build the intended neural network. Commonly, a full connection between the neurons of two adjacent layers, as shown in Fig.5, of a feed-forward network is utilized. Unused connections can be deactivated through the training process by application of a zero weight factor.

In a feed-forward network we can identify three different layers. In the input layer the external stimulation of the network will be adapted to the requirements of the network. The real processing takes place in the hidden layer. This can consist, as shown in Fig.5, of one layer or, for more extensive networks, of several layers. At last, in the output layer the results of the network will be transformed for the external representation.

### 3.1.1 Classification

In the original implementation of the neuron the activation function decides whether or not the neuron is activated. This leads to a leap function for the output and maps the possible outputs to two discrete values. In a multilayer network, the weight multiplier of the following neurons can be simplified for such an output signal. Unfortunately, this disturbs the calculation flow in the network and reduces also the information about the strength of the activation. Therefore, mostly non-discrete activation functions are used.

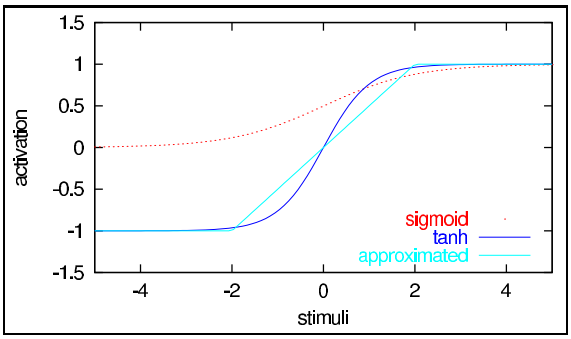


**Figure 6:** Plot of possible activation functions

For the implementation with digit online arithmetic we can simplify the calculation of the activation function with an approximation as shown in Fig.6. This new function for the classification can be implemented by a simple multiplication followed by a limitation. Through the utilized MSD-first data flow this can be easily done without any speed penalty.

### 3.2 Working Phases

Due to the different tasks we must differentiate two working phases.

#### 3.2.1 The Evaluation Phase

From the user's point of view is this the actually required work mode. The network determines based on the trained behavior an evaluation of the presented input dates. In a feed-forward structure this means the gradual processing through the network layers without any feedback data-paths. Consequently, the circuit structure for the evaluation phase can also be implemented as a pipeline.

#### 3.2.2 The Training Phase

A training is only necessarily because of the expenditure to stores the weights. Further, the possibility of the training increases the flexibility of the network applicability. We only have to change the training patterns to adapt the network to a new task. The training itself can be done by the same learning algorithm as before.

Alternatively, for a single application it is also possible to omit the on-chip learning phase and to fix the weightings in the circuit. In this case, you need a new chip design for each small modification within the training samples.

## 4 The Network

The presented network is the second step in the verification of our implementation concept. We start with the implementation of a network which solves the academical XOR problem through a neural network. The resulting network consist of only five neurons, two in the input layer, two in the hidden layer and one for the output layer. Due to the small size of the network we had no problems with the verification of the correctness of our design.

After that we were looking for a more practical application. We decided to implement a waveform detection circuit, which is also a somewhat academic problem by substantially increased in their resources requirements.

A second goal of this project was the development of a reference design for our dynamic circuit technology design-flow. With this project we want to prove the universal applicability of our design-concept.

### 4.1 The Concept

We defined the task of the neural network for waveform detection as: any presented waveform should be classified into one of three possible categories: a sinus, a rectangle or a triangle waveform. To simplify the network architecture, we decided to present not only the actual value of the amplitude but also the last n values. By this decision we can also utilize a simple feed-forward network structure like in the XOR-example.

Because of the selectable accuracy for the internal value representation we can also investigate the influence on the evaluation performance and the learning behavior of the network. Commonly, an accuracy between 8 and 16 digits is usable.

As a further scalable design parameter we can vary the expansion of the hidden layer. We limited our design to only one layer in the hidden area to speed up the performance of the network.

### 4.2 The Matlab Model

First, as already done with the development of the XOR network model, we designed a network description for the Matlab environment. With this model we verified the correctness of our network model and the ability to solve the design problem with it. This model was also very helpful for the sizing of the hidden layer network.

During the design of the VHDL model of the network we also used the Matlab model to generate stimuli for the verification of parts of the network. This way we tested the evaluation phase of a small network piece before we developed the learning component with weightings derived from a Matlab simulation.

### 4.3 The Simulation Test

After the successful verification of network parts we tried to simulate the whole waveform detection network. At this point of the design state we run into trouble with the abilities of our simulation tools.

In the first attempt we started a simulation of the plain VHDL sources with the VSS simulator. As already seen during the simulation of the XOR network, the simulator has difficulties with the extensive use of the generic Takagi adder table description, which permits the variation of the SD-encoding style. We were not able to start a simulation run, because the simulator could not finish, even after several days, the binding of the simulation files. So we changed our VHDL source to a fixed SD-encoding and a combinatoric Takagi adder description with the same result. Only after we reduced the hidden layer to a small size, not sufficient for our problem, our simulation started after many hours of preparation time.

With the cyclone simulator we had the same problems so that we had to find a solution for our verification problem.

### 4.4 The Synthesis

Due to the experiences with the simulation we decided to use a hierarchical synthesis approach.

At first we analyzed and elaborated the whole network. Then we synthesized the basic SD adder structures and marked them as “don’t touch”. Afterwards we processed the basic digit on line blocks for the base operations like multiplication and the activation function. These steps are followed by the compilation of the different base neurons for the input, hidden and output layer. The last task for the Synopsys Design Compiler was the processing of the whole network structure.

For all compilations we use an area limiting synthesis script. The final place and route step was completed with the XILINX M1.5 tool set.

### 4.5 The Aptix Verification

Thanks to the friendly support by the DFG, the German research foundation, since last year we are able to use a System Explorer MP3C hardware verification system (Fig.7) from Aptix. This system equipped with four XILINX VIRTEX1000 devices give us the possibility to build up a hardware based verification of our network. Unfortunately, the interconnection of the board to the simulation through the MVP product was not available due to the limited financial resources. So we

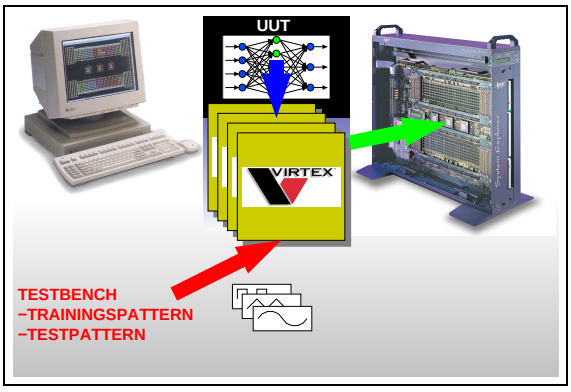


**Figure 7:** Aptix MP3C system

had to implement our synthesizable test-bench in one of the VIRTEX devices on board. Also, the test and training patterns could be accommodated in the test-bench fpga by utilization of the internal ROM structures.
The partitioning of the network on the remaining three fpga could easily be managed because of the small degree at connections between the individual neurons.

### 4.6 The Transformation to dynamic Implementation

In the last task of this design-project we will currently transform the design into a dynamically representation. For the cell library development we use the so called TSPC circuit style first presented in [YKS87]. Due to the dynamic character of the circuit technology we have slightly modified our standard design-flow as shown in Fig.8.
After the translation to the dynamic cell library we will perform a second verification run on the MP3C system before we start to generate the final layout of the design. Therefore, we have developed an appropriate VHDL description of the dynamic cells, which can easily be mapped on the VIRTEX devices due to the internal structure.

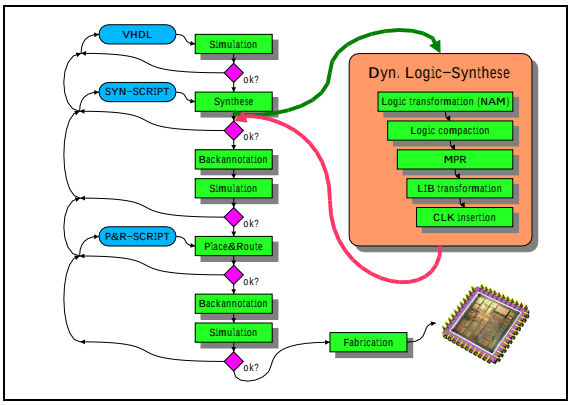


**Figure 8:** TSPC Design-Flow

This additional verification gives us the possibility of the examination of the complete pipeline implementation.

# 5 Results of verification

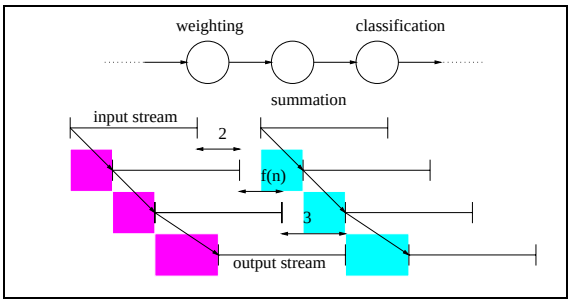


**Figure 9:** Analysis of the data-flow in the evaluation phase

As shown in Fig.9, the throughput of the network is mainly determined by the highest digit online delay $\delta$. The throughput is influenced by the observed growing of the vector length in some operations like the digit online addition. To reduce this effect the implemented activation function of each neuron should also carry out a limitation of the digit vector length.

Furthermore, synthesizing single neuron architectures with different parameters for input vector count and calculation accuracy produces the following results (Tab.2).

The high io-count in Tab.2 is determined by the connection for the external weight observation in the debugging version of the network.

| input vec. | bit width | clb slices | func . gen. | flip flops | io pins | gate equ. |
|---|---|---|---|---|---|---|
| 1 | 8 | 462 | 921 | 426 | 53 | 9648 |
| 1 | 16 | 567 | 1131 | 474 | 101 | 11292 |
| 2 | 8 | 702 | 1403 | 671 | 71 | 14752 |
| 2 | 16 | 843 | 1683 | 735 | 135 | 16944 |
| 5 | 8 | 1220 | 2435 | 1175 | 125 | 25480 |
| 5 | 16 | 1465 | 2925 | 1287 | 237 | 29316 |
| 10 | 8 | 2223 | 4431 | 2169 | 215 | 46416 |
| 10 | 16 | 2646 | 5271 | 2361 | 407 | 52992 |
| 20 | 8 | 4086 | 8147 | 4003 | 395 | 85232 |
| 20 | 16 | 4857 | 9687 | 4355 | 747 | 97288 |
| 50 | 8 | 9260 | 18467 | 9043 | 935 | 192512 |
| 50 | 16 | 11082 | 22107 | 9875 | 1767 | 221008 |

Table 2: Synthesis results for a single neuron

# 6 Conclusion

The presented solution for the hardware based verification of a neural network design example demonstrates a work around for verification problems caused by the potentially expansion of the implementation efforts on scalable architectures.

The embedded test-bench is applicable but lacks on comfort during the debugging of the design. For each minimal change on the test-bench we had to perform the whole fpga/Aptix-design-flow.

So the connection of the verification hardware to a software based test-bench simulation through tools like the MVP is desirable.

# References

[Erc84] M.D. Ercegovac. On-line arithmetic: an overview. *Real Time Signal Processing VII*, 495:86–93, 1984.

[ET77] M.D. Ercegovac and K.S. Trivedi. On line algorithms for division and multiplication. *IEEE Transactions on Computers*, C-26 No 7:681–687, July 1977.

[FS91] J.A. Freeman and D.M. Skapura. *Neuronale Networks: ALgorithms, Applications and Programming Techniques*. Computation and Neural Systems Series. Addison-Wesley Puplishing Company, 1991.

[KET87] S. Kuninobu, H. Edamasu, and N. Takagi. Design of high speed mos mul-

tiplier and divider using redundant binary representation. In *Proc. 8th Symp. Computer Arithmetic*, pages 80–86, New York, 1987.

[RR91] U. Ramacher and U. Rückert. *VLSI Design of Neuronal Networks*. Kluwer Academic, Boston, 1991.

[Sma92] C. Smalla. Untersuchung, Entwicklung und Simulation von rekonfigurier- und pipelinebaren Digit-Online-Verfahren für die digitale Signalverarbeitung. Technical report, Universität Duisburg, 1992.

[YKS87] J. Yuan, I. Karlsson, and C. Svensson. A true single phase clock dynamic cmos circuit technique. *IEEE Journal of Solid State Circuits*, SC-22:899–901, 1987.

[Zel97] A. Zell. *Simulation neuronaler Netze*. R. Oldenburger Verlag, München, 2 edition, 1997.