Time Coding Output Neurons in Digital Artificial Neural Networks

Ralf Joost, Ralf Salomon
Institute of Applied Microelectronics and Computer Engineering, University of Rostock
Rostock, Germany
ralf.joost@uni-rostock.de

***Abstract*-Previous research has proposed several platforms for the direct hardware implementation of neural networks. Most of these platforms employ a small number of special-purpose processors that emulate a given number of neurons. By contrast, this paper proposes a *light-weight* hardware model, called *time coded output* (TiCO) neuron, which codes its internal activation as the duration of its output pulses. Since the implementation of such a neuron requires only about 200 logical elements, standard state-of-the-art field-programmable gate arrays may host several hundreds of neurons on a single chip. Due to its output coding and its low footprint, TiCO neurons are particularly suited for the usage in embedded systems.**

Keywords: neuron design, hardware implementation, FPGA, pulse width modulation

I. INTRODUCTION

No doubt, neural networks of various sorts are indispensable tools in research and development. From an application point of view, the advantages of artificial neural networks are their massively parallel processing nature as well as their capabilities to learn from examples and to generalize previously unseen data. Last but not least, artificial neural networks are able to provide parameterizable and very compact models of complex, high-dimensional, multi-modal functions.

In way contrast to their biological role models, most artificial neural networks are implemented on traditional, rather serial processors, such as PCs and micro controllers; parallel processing is a rather secondary property that emerges if the networks is partitioned across a few "regular" processing cores. In terms of processing speed, a PC-based implementation constitutes a significant computational bottleneck: if a multilayer backpropagation network would consist of 10,000 neurons, even a 10 GHz processor would be able to calculate only about 20 forward paths per second.

In light of the discussion presented above, the use of *dedicated* hardware platforms for artificial neural networks seems rather attractive particularly for high-performance applications. Section II provides a brief overview of previous research in that area. This review reveals two prevailing trends: first, analog hardware with limited computational precision and scaling properties, and second, coarse-grained multi-core platforms that employ a moderate number of heavy-weight, serialized multipliers.

Hardware chips known as field-programmable gate arrays (FPGAs) are particularly interesting for neural network implementations: the (application) programmer can configure, place, and interconnect all the various logical gates at his or her own disposal. With an FPGA, a programmer can follow the traditional approach of constructing hardware multipliers and (application-specific) ALUs. However, FPGAs also allow for *any* interconnection, and thus for any form of (digital) hardware implementation.

Thus, Section III proposes an alternative implementation approach for artificial neurons, called the *time coded output* (TiCO) neurons: rather than coding a neuron's activation level as a single floating-point value, TiCO neurons communicate their signals with a duty cycle that is proportional to a neuron's internal activation state. Because of this property, TiCO neurons are particularly suited in embedded systems, which are normally tightly coupled with some physical processes.

Practical experiments have demonstrated the feasibility of implementing TiCO neurons on standard FPGAs. Section IV provides all the technical details and algorithms used in these experiments. Then, Section V summarizes the achieved results, which indicate that off-the-shelf FPGAs allow for parameterizable and dynamically modifiable duty cycles on oscillating signals. Section VI discusses the achieved results in terms of performance, resource usage, and application issues.

The TiCO neurons proposed in this paper are a first step towards an alternative neuronal network implementation, which tries to preserve the inherent parallel processing capability of most of the existing artificial network models. Section VII discusses some of the remaining research issues, and outlines the next steps of future research.

II. BACKGROUND: HARDWARE IMPLEMENTATION PLATFORMS

A. Digital Implementations of ANNs

Most neural network models process their inputs in several stages (see, also, Fig. 1). Multilayer perceptrons [14], for example, first calculate their net input *net* according to the assigned input values I_j and link weights w_j for all n inputs

$$net = \sum_{j=0}^{n} w_j \cdot I_j. \quad (1)$$

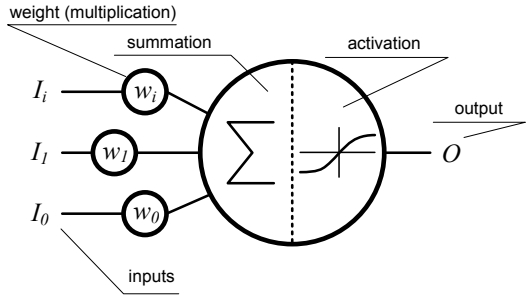

Figure 1. The multilayer perceptron

Then, they translate their net-input to the output value *o* by using some nonlinear transfer function, such as the logistic function

$$o(net) = \frac{1}{1+e^{-net}} \quad (2)$$

or the hyperbolic tangent

$$o(net) = \tanh(net). \quad (3)$$

In the most straight-forward way, a digital implementation employs a set of heavy-weight processing units. Fig. 2 shows a generic hardware architecture of a perceptron. As can be seen, the artificial neuron consists of three stages: (1) input-weight-multiplication, (2) accumulation, and (3) output value computation. The size of the shown weight storage RAM depends on both the number of connected inputs and the chosen precision. Previous research [2] has recommended that at least a 16-bit precision should be used. Thus, a 16-bit hardware multiplier is necessary to calculate a 32-bit partial product per clock cycle. The result is stored in a temporary product storage and accumulated in the subsequent clock cycle. Given *n* input values, the accumulation of all weighted input values requires *n* clock cycles. Once the final sum is calculated, the output value of the sum storage register addresses an entry in the activation function look-up table (LUT). This means that the activation function is pre-calculated for a given input range and a given precision. Muthuramalingam et al. [3] have present an example for this approach using an FPGA-implemented neural network for space vector modulation. According to Muthuramalingam et al., the main difficulties are the trade-off between parallelism and resource requirements as well as the intended precision. They indicate that the utilization of look-up tables for the activation function saves approximately 70 percent of resources compared to a full computation approach. However, a three-input neuron still requires 25 clock cycles, and more than 560 logic elements[1] to do one output calculation.

For simple threshold neurons, previous research [4] has proposed an alternative that relies on AND and OR gates. The main benefit of this approach is that it results in a rather simple hardware structure, whereas a disadvantage can be seen in the

[1] In this paper, a logic element is defined as a combination of a four-bit input look-up table and a flip flop. The term "slice", originally used in the referenced literature, is vendor specific (XILINX) and calculated as two logic elements. Since the introduction of the Virtex 5 FPGA family, a "slice" contains four logic elements.

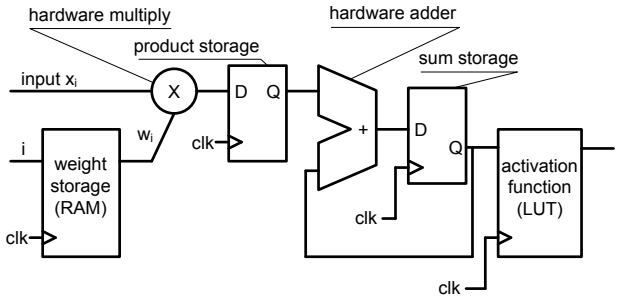

Figure 2. Generic hardware architecture for an FPGA-implementation of a perceptron

limited number of input values. According to the authors, the complexity of their approach grows exponentially in the number of input bits, and thus may give preference to the classical computation approach.

The usage of 16-bit floating-point values has aready been investigated by previous research [5]. Unfortunately, the implementation of a 2-2-1 network for the well-known XOR-problem requires already approximately 9800 logic elements.

Further examples for FPGA-implemented neural networks can be found in the pertinent literature, where particular interest is given to the domain of industrial control applications [6, 7].

B. *Analog implementations:*

Some existing analog implementations avoid digital processing units by operating all the CMOS transistors in their non-linear, non-saturation regime. On the one hand, analog systems offer the placement of a large number of high-speed neurons in silicon. On the other hand, analog circuits are known to suffer from noise, interference and process variations.

Stüpmann [11] has presented a concept and a detailed layout description for an analog integrated circuit, implementing a neural network capable of learning [11]. Weight multiplications as well as the sigmoidal activation function were realized by a Gilbert multiplier. Furthermore, capacitive elements provided the required weight storage functionality. However, the design was far from a mass-production state.

Figueroa et al. [1] have pointed out that in analog circuits the main issues are nonlinearities in the current-to-voltage transfer characteristics. To provide a tool for predicting implementation performance, they have designed an FPGA-based network emulator in order. The emulation required the transfer of both analog multipliers and memory cells to the digital FPGA world. In the end, they used a concept, named temporal synapse slicing, to keep the number of required FPGA resources low. Again, mixing the analog behavior of neurons and digital processing is interfered by the requirements of the multiplications and transfer functions.

III. THE TICO NEURON

This section presents a different approach for reaching analog behavior without leaving the digital world. This approach is named *Time Coded Output Neuron* (TiCO neuron).

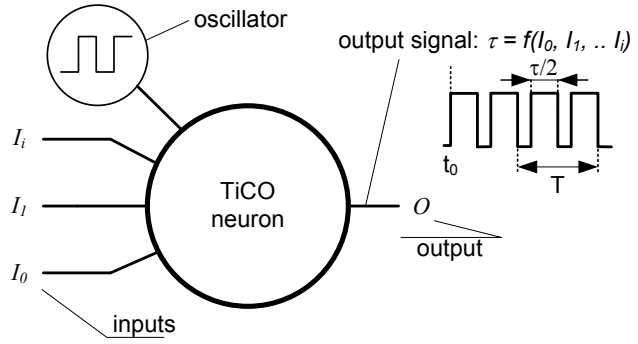

Figure 3. TiCO: the general mode of operation

This neuron is characterized by the use of the pulse width modulation to encode analog information on a single bit signal line.

A. Overview

The goal of the TiCO neuron is to provide a simple, low-cost implementation variant for artificial neurons. This concept is illustrated in Fig. 3. The basic idea is to offer neurons that emit pulses with variable lengths. The pulse length $\tau = f(net)$ is proportional to a neuron's activation value, and thus directly codes for its output state. In other words, a TiCO neuron is a voltage-controlled oscillator (VCO) in which not the frequency but the duty cycle is the target.

B. Generating pulses with variing duty cycles

To generate rectangular signals, most integrated circuits utilize some form of dedicated hardware, such as phase locked loops or other types of oscillators. The signal's parameters are its frequency *f*, its period $T=1/f$, and the signal's duty cycle τ. Usually, the duty cycle $\tau = T/2$ is half the signal's period *T*. In order to obtain a duty cycle that differs from the value $\tau = T/2$, the designer has two options. The first option would be to change the properties of the global oscillator, which would be, however, quite useless, since the duty cycle has to be adapted for every single neuron on an individual basis. Therefore, the duty cycle has to be changed *within* every neuron. This can be done in the following way: the oscillator signal *s(t)* is duplicated to *s'(t)*. This duplicated signal is phase shifted to $s'(t-\Delta t)$. Finally, both signals *s(t)* and $s'(t-\Delta t)$ are combined by an exclusive-or gate XOR($s(t)$, $s'(t-\Delta t)$). This procedure is exemplified in Fig. 4. Valid delay values Δt are in the range from *0* to *T/2,* since other values introduce some form of unambiguity due to the signal's periodic behavior. The duty cycle $\tau = 2\Delta t$ is twice as long as the applied delay value Δt.

C. Delay generation

The pertinent literature offers at least three different methods to generate delays in integrated circuits. The classical approach is to use the processing delays of the FPGA's logic elements [10]. These delays depend on the FPGA technology, and are usually in the range of 200 ps to 400 ps. Thus, a delay chain with a resolution of 200 ps to 400 ps can be obtained by serially connecting any desired number of logic elements. A more fine-grained delay chain can be implemented by using the special-purpose carry path between adjacent logic elements.

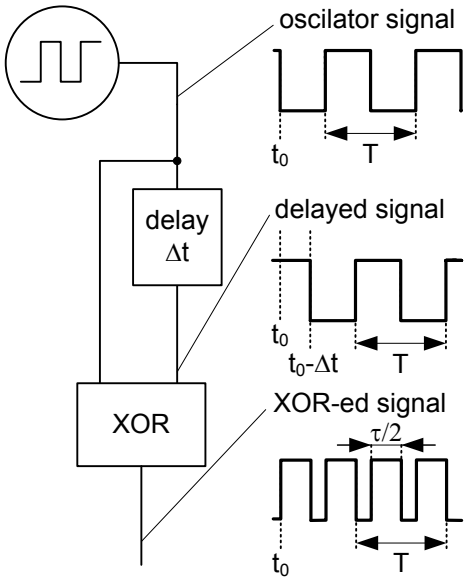

Figure 4. General approach to generate variable duty cycles

This carry path is usually dedicated to fast ripple-carry adders, and yields a processing delay of approximately 30 ps [17]. It should be kept in mind, though, that the use of logic elements as parts of a delay chain implies two disadvantages. First, the use of logic elements increases the design-size significantly, slowing down all involved development processes, i.e., synthesis, placement, and debugging. Second, logic elements tend to vary their processing speeds (delays) according to the exogenous parameters, such as the on-chip temperature and the on-chip supply voltage setting [12].

Recent research has suggested a third way of generating on-chip delays. The BOUNCE/X-ORCA architecture [8, 9] is a high-performance time measurement system that yields a resolution of better than 10 ps by utilizing the propagation delays that are imposed by an integrated circuit's passive electrical wires. Due to the regular structure of an FPGA, it is possible to place two logic elements such that the connecting signal between both exhibits any desired delay. In other words, increasing the geometrical distance between two logic elements that are placed on the FPGAs regular grid also increases the delay between those elements. Only the geometrical size of the FPGA limits the maximum segment length of a signal wire, and thus the achievable delay between two logic elements. However, consecutively connecting multiple instances of these delay segments again ensures a wide range of achievable delays. This approach has two benefits: the number of used hardware resources remains low and the delays are apparently more stable against temperature changes.

D. Implementing the TiCO neuron

A single TiCO neuron is achieved by connecting a user-defined number of simple TiCO building blocks as shown in Fig. 5. In essence, every TiCO building block is able to shift the incoming oscillator signal by a dedicated delay Δt. Furthermore, the neuron's input vector is used to control whether a TiCO building block feeds the delayed signal or the

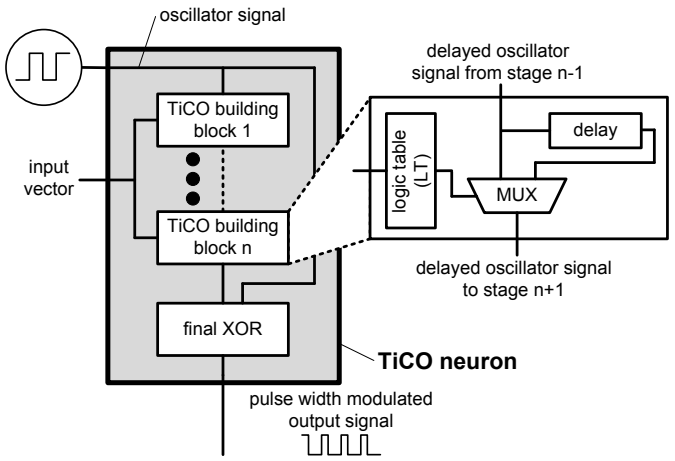

Figure 5. The Basic Design of TiCO neurons relies on the use of multiple TiCO bulding blocks

un-delayed signal to the next TiCO building block. The generation of this control signal is performed inside every single building block by a logic table (LT). The generated control signal is only valid for that block. Thus, the user may define different control functions for every block. As a consequence, a particular input vector may enable the delay of some building blocks whereas others transfer an un-delayed signal. In so doing, the entire system of TiCO building blocks results in an overall phase shift. Finally, the phase-shifted signal is XOR-ed with the un-shifted original signal, producing a pulse width modulated signal with a duty cycle of 0% to 100%.

As can be seen in Fig. 5, delays are only imposed on one channel; the original oscillator signal bypasses all delay stages and feeds the final XOR. Since already the latency of the integrated multiplexer introduces a delay to the signal, the original oscillator signal also has to pass the very same number of multiplexers to correct this effect. These bypass multiplexers are not shown.

The delays should be designed such that the overall phase shift is in the range of 0° to 180°. This requires the designer to know the incoming oscillator frequency and to adjust the delays in every stage accordingly. The simple structure shown in Fig. 5 allows only for positive phase shifts. In every stage, the phase shift is increased or remains unchanged. Since artificial neurons also accept negative weighted values, leading to a decreasing net input, the design misses full functionality. However, if the very same structure is applied to both signal paths, virtually positive and negative phase shifts are provided.

E. Resource usage

The number of resources used in the TiCO building block is the chosen metric, which depends on several parameters. The number of used input bits determines the number of LEs to generate the look-up table. If 2^n denotes the number of input bits, $(2^n/2-1)$ logic elements are required for $n>1$. The number of logic elements used as buffers to form the signal delay depends on both, the chosen oscillator frequency as well as the geometrical chip size. The lower the frequency the higher the delay values necessary to gain a full 180° phase shift. Also, the smaller the chip's size, the more buffers have to be used to form a zick-zack wire layout pattern that acts as a multi-segment delay path. However, at least one logic element is required. In addition, the multiplexer is realized with the help of one logic element. In conclusion, a four-input bit TiCO building block utilizes at least three logic elements. The answer on the question of how many TiCO building blocks form one TiCO neuron is problem/design specific. For example, a TiCO neuron with four TiCO building blocks and a four-bit input vector allows for 16 linearly distributed phase shifts and consumes at least 12 logic elements. However, this can be seen as the absolute minimal requirement, practically useful neurons may utilize 100 to 200 logic elements or even more.

IV. Methods

A. FPGA Specification

All experiments were done with an Altera Stratix III development board. This board is equipped with an EP3SL150 FPGA chip that provides approximately 150,000 configurable logic elements. Altera's Quartus 11.0 design suite was used for synthesis. The free running oscillator signal is provided by one of the four internal phase-locked loops. The signal frequency was set to 3 MHz. To allow for an external signal inspection, the generated output is monitored by a Tektronix TDS 2024 oscilloscope. To provide a proof-of-concept, two TiCO neurons with different transfer functions were realized.

B. Linear Transfer Function

One of the simplest transfer functions is of linear shape. This means that the resulting phase shift between the delayed and the un-delayed oscillator signal linearly depends on the input value. For this experiment, a neuron with four input bits was chosen. Although relying of just one four-bit input vector is of limited practical relevance, it serves the purpose of showing the hardware-related details. Obviously, any different number of input values can be used in the very same way.

Using four input bits allows for 16 distinguishing phase shift values. As already said, the oscillator signal was set to a frequency of 3 MHz, or a pulse width of 333 ns respectively. To achieve a phase shift of 180 degrees, a maximum signal delay of 166 ns has to be implemented. Thus, four different TiCO building blocks were used with every single one responding to one input bit. The following four delays were chosen to 11 ns, 22 ns, 44 ns, and 88 ns. In case all the four input bits are activated, an overall phase shift of approximately 166 ns is applied to the delayed oscillator signal.

C. Logistic Transfer Function

Since the logistic transfer function is of special interest in neural networks, a second experiment has generated delays in a TiCO neuron such that the dependency of the pulse width modulated signal's duty cycle in relation to the input values follows a sigmoidal curve. This time, a six-bit wide input vector was chosen. The 64 possible input values were mapped onto a sigmoidal curve in the range from -4 to 4.

In this experiment, the oscillator's signal frequency was set to 4 MHz, requiring a maximum delay of 125 ns for a 180 degree phase shift. 64 TiCO building blocks were used to realize the TiCO neuron. In every block, the logic table acts as

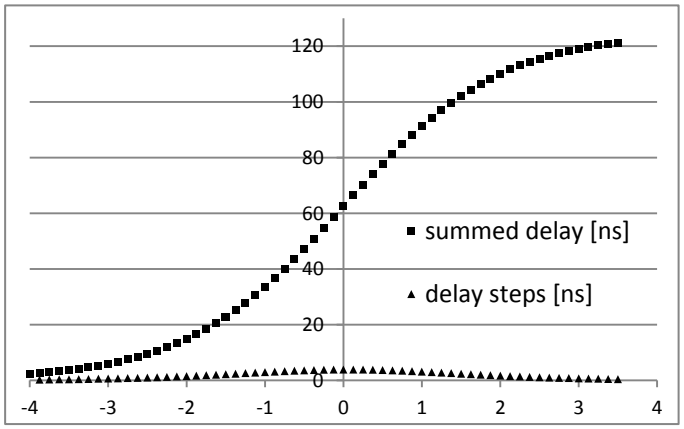

Figure 6. Mapping of a 125 ns delay to a sigmodial curve and resulting delay steps for threshold switches

a threshold switch. The delay of each block is switched on, when the input value is greater than the threshold of that block. In so doing, the final delay is achieved by summing up the delays of various blocks. This keeps the delay that has to be generated in every block rather small. Fig. 6 shows both the summed delay values as well as the delay steps that are generates inside the 64 building blocks.

V. RESULTS

Visual inspection of the output signals done with the help of the aforementioned Tektronix oscilloscope indicates that both neurons' behaviors fulfill the theoretical expectations. Furthermore, the placement of the TiCO building blocks, especially the generation of delayed signals, allows for automation, since all coordinates can be included in simple text assignments. The Quartus synthesis tool, used in the practical experiments, includes a timing analyzer that calculates the expected delays according to a given configuration. Basically, this enables the tuning of all delays by some software approaches, such as genetic algorithms.

A. Linear transfer function

The chosen Stratix III FPGA generates delays of approximately 2.2 ns when a signal is routed horizontally across the entire chip. Thus, to realize longer delays, the delay path crosses the chip multiple times. Every time, the delay path changes its direction, an additional logic element that acts as a buffer is required. This additional element also introduces a certain delay that increases the overall delay on that path. However, by placing the buffers on the chip, any desired delay is realizable. Placing the elements is simply done with the help of the localization assignments in form of x- and y-coordinates.

A timing analyzer [13] is integrated in Altera's development tools and gives first hints about the generated delays. The timing analyzer is able to apply different timing models to the synthesized hardware. Those models contain all delay information for the FPGA's physical components, such as logic elements, memories, and signal wires. The designer can choose between models that differ in the assumed operation parameters, such as the core voltage, the core temperature, and the chip's speed grade. Depending on the model, the analyzer calculates the expected delays on the hardware's data paths. For this particular experiment, Table I shows the number of chip crossings as well as the expected delays calculated by the timing analyzer tool. For timing analysis has utilized the "fast" and the "slow" timing model for the Stratix III FPGA as provided by Altera

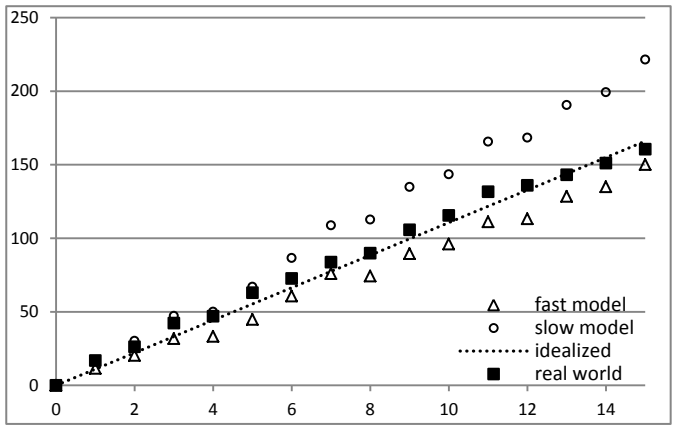

Figure 7. Delay values in nanoseconds for the linear transfer function. The real-world delays lead to phase shifts between 0° and 180° for a 3 MHz signal

TABLE I. TIMING AND PLACEMENT INFORMATION FOR TICO BUILDING BLOCKS USED IN THE LINEAR TRANSFER FUNCTION EXAMPLE

Block number	Chip crossings of delay signal	calculated delay (fast/slow)
(1)	6	15,2 ns / 22,2 ns
(2)	12	21,7 ns / 30,8 ns
(3)	22	38,9 ns / 55,7 ns
(4)	42	74,3 ns / 112,7 ns

The neuron implementing the linear transfer function consumes 110 logic elements although only four input bits were used. Fig. 7 shows the phase shifts for the 16 different input values measured with the Tektronix oscilloscope. As can be seen, the finally achieved real-world delays differ from those delays that were calculated by Altera's timing analysis tool. This indicates that an automated delay adjustment should rely on real-world data rather than on calculated delay values. This aspect is discussed in detail in Section VI.

B. Logistic transfer function

As in Section IV announced, a neuron with a logistic transfer function was realized. Here, the single TiCO building blocks uses even less resources compared to the linear transfer function. As can be seen in Fig. 6, the maximum delay step does not exceed 4.5 ns. Thus, it is sufficient to route a single delay path just across the chip and back within one building block. In combination with the necessary buffer at the turning point of the delay path, a maximum delay of 4.7 ns can be realized. In consequence, the single TiCO building block requires just one logic element for delay generation.

The resource usage for the entire neuron implementing the logistic transfer function is quite low: the synthesis tool states an overall usage of only 212 logic elements for the 64 building blocks. For this experiment, the building blocks as well as the delay paths were placed manually on the chip. The assumption was that especially the delay of a signal path linearly depends

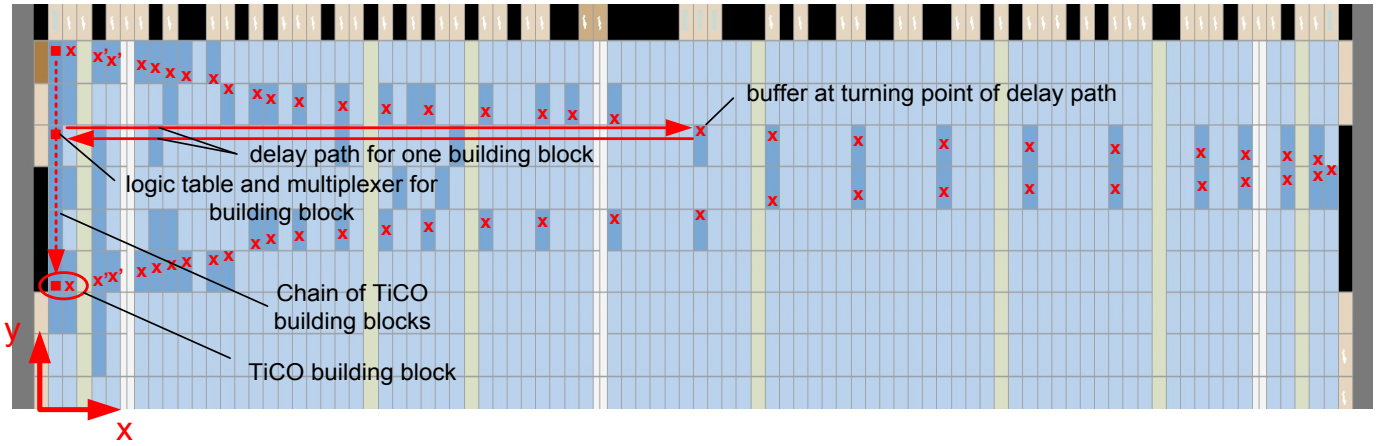

Figure 8. Illustration of on-chip placement for one TiCO neuron. The TiCO neuron consists of multiple TiCO building blocks. Every building block contains a logic table and a multiplexer (illustrated by a square) and a delay path realized by dedicated signal routing to a buffer (illustrated by 'x' – the symbol x' indicates multiple buffers in this area). Picture taken from Altera's chip planner tool, illustrations added to improve visibility.

on the signal wire's length. Thus the logic tables and the multiplexer, shown in the schematic (Fig. 5) were placed on the left-hand-side of the chip, all delay signals were routed to the right side. The final resource placement on the chip is illustrated in Fig. 8. As can be seen, the buffers at the turning points of the delay path form a shape, comparable to the delay steps in Fig. 6.

Fig. 9 shows the phase shift distribution according to a given input value for the logistic transfer function. As can be seen, the calculated delays (marked by circles) from the timing analyzer fit the expected sigmoidal function. Unfortunately, the real-world results exhibit smaller delays than expected. After recognizing this effect at selectively chosen test points (marked by solid triangles), the placements of the buffers of all TiCO building blocks were manually adjusted. The adjusted delays are marked by the 'x'-symbol in Fig. 9.

VI. Discussion

To proof the concept of the TiCO neurons, only simple neurons were implemented on the FPGA during first experiments. The results shown above indicate that on-chip

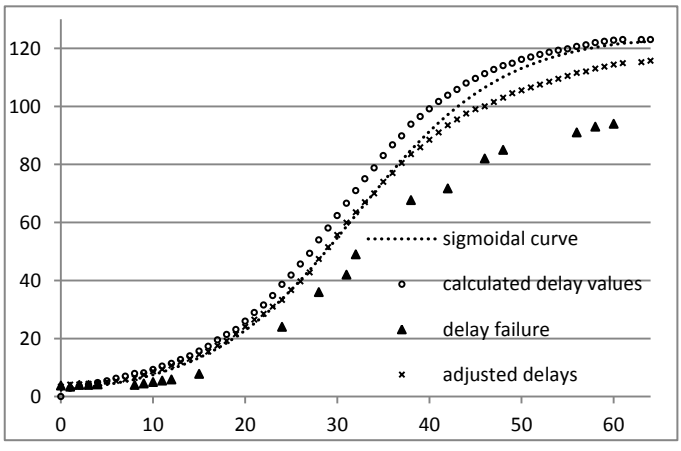

Figure 9. Delay values in nanoseconds for a sigmoidal shaped phase shift according to a given input value. The delays yield phase shifts between 0° and 180° on a 4 MHz signal

signal delays can be adjusted to any desired behavior. However, a point of concern is, whether or not the TiCO neuron is able to deliver output values at a resolution that is sufficient for complex network structures. Here, two different aspects have to be taken into account.

The first one is the frequency of the utilized oscillator signal that is delayed inside the TiCO building blocks. As indicated, two different frequencies were used, 3 MHz for the linear transfer function and 4 MHz for the logistic transfer function. To realize a phase shift of 180 degrees, delays of 166 ns (3MHz) and 125 ns (4MHz) are necessary. If one assumes, that differences in the pulse widths as low as 1 ns are detectable, these signals allow for 125 (166) different values. To increase the number of distinguishable values, the oscillator frequency has to be scaled down. On the other hand, a lower frequency yields in higher resource usage due to the required longer delays. Here, an optimal tradeoff depends on the problem at hand.

The second aspect is the minimal achievable delay in one delay path. Since the TiCO building block (see, Fig 3) requires one logic element within the delay path, the delay is at least as long as the latency of that logic element. That latency depends on how the synthesis tool configures the logic table of that logic element. The results indicate that values between 0.077 ns and 0.380 ns are possible. Up to now, synthesis was performed automatically; future research will target at low latency configurations in all logic elements. However, at least 0.077 ns are introduced by a delay line. A possibility to overcome this step and to allow for finer-grained delays might be the inclusion of further logic elements for compensation. Such compensational logic elements are already used inside the un-delayed signal path of the TiCO building block, since the signal's passing of the multiplexer inside the building block already introduces an undesired delay. However, the approach of compensational logic elements requires additional care due to the requirement of equally formed delays.

Another question concerns the number of how many input bits may or should be processed by a single TiCO neuron. The examples discussed in Section IV used input vectors with four

and six bits. Of course, the concept of the TiCO neuron allows for higher numbers of input bits. Since the shown implementations were manually placed, the number of input bits was kept small. To handle more complex neurons, the generation and placement of the neuron's logic elements has to be done automatically or at least with computational support.

The resource usage of the logistic transfer function depends exponentially on the number of used input bits, since for any possible input value, the sum of subsequent delay steps is formed. The impact of that method can be lowered, if adjacent input values are grouped and just one delay value for that group is provided. Again, an optimal tradeoff depends on the desired resolution and the utilized oscillator frequency.

Due to its internal structure, a TiCO neuron has a significant performance advantage: Since no complex computation is performed, the worst-case latency of the TiCO neuron is less than or equal to the maximum delay that is realized.

VII. Conclusion

The results of the shown experiments indicate that of-the-shelf field-programmable gate arrays are able to provide stable, manually adjustable delay paths, which may be used to implement artificial neurons. These neurons deliver an output value that is a pulse-width modulated signal, where the pulse width modulation can be adjusted to any desired behavior. Here, the logistic transfer function may be of particular interest.

Future research will be dedicated mainly to the following three topics. First, an approach for the automatic delay adjustment has to be implemented. This includes the automatic detection of the realized delay values, which differ from those delays that are provided by the timing analyzing tools.

Second, research will be dedicated to the realization of fully functional TiCO networks. Here, it is not only necessary to place multiple instances of these neurons on one chip but also to find a way of using TiCO neurons in all layers. For complex artificial networks that solely consist of TiCO neurons only, research has to be dedicated towards retranslating pulse width modulated signals back into digital data. For this task, counters are a promising option. If done so, the TiCO neuron enables fully parallel networks with a continuous mode of operation.

Third, the automatic online adjustment of the TiCO neurons is a future research goal. Currently, placement has to be calculated on a PC. Synthesis and FPGA configuration also requires a PC. During the configuration phase, the entire FPGA is halted and reprogrammed. However, state-of-the-art FPGAs, namely those manufactured by XILINX, enable dynamic reconfiguration. This approach allows for partially reprogramming the FPGA without the need of complete shut-downs. Here, only the reprogramed part is turned off, reconfigured, and turned on again. The rest of the chip's hardware just operates as usual. This may allow for dynamic changes in the implemented network without any stopping operation.

In conclusion, the approach of the TiCO neuron may lead to a new variant of hardware implemented neural networks. Especially the low resource usage of these neurons may allow for complex net structures. As shown in Section V, a TiCO neuron with six input bits implementing a logistic transfer function requires just 212 logic elements. Already today, Altera's Stratix V GX FPGA provides more than 700,000 logic elements, XILINX' Virtex 7 ships even more than 1.2 million logic elements[2].

Acknowledgment

The authors gratefully thank Mathias Hinkfoth, University of Rostock, for his support during the project. Special thanks are due to Prof. Leon Glass for the fruitful discussion he had with Ralf Joost during the IJCNN conference 2011. This discussion has significantly influenced the presented research.

References

[1] M. Figueroa, E. Matamala, G. Carvajal and S. Bridges, "Adaptive signal processing in mixed-signal VLSI with Anti-Hebbian learning", Emerging VLSI Technologies and Architectures (ISVLSI 06), pp. 6-11, 2006

[2] J. Zhu and P. Sutton, "FPGA implementations of neural networks - a survey of a decade of progress", Lecture Notes in Computer Science, vol. 2778/2008, pp. 1-4, 2003

[3] A. Muthuramalingam, S. Himavathi and E. Srinivasan, "Neural network implementation using FPGA : issues and application", International Journal of Information and Communication Engineering, vol. 4, no. 6, pp. 396-402, 2008

[4] A. Dinu, M. N. Cirstea and S. E. Cirstea "Direct neural-network hardware-implementation algorithm", IEEE Transactions on Industrial Electronics, vol. 57, no. 5, pp. 1845-1848, 2010

[5] M. A. Çavuşlu, C. Karakuzu, S. Şahin and M. Yakut, „Neural network training based on FPGA with floating point number format and it's performance", Neural Computing and Applications, vol. 20, no. 2, pp. 195-202, 2010

[6] H. H. Lee, "The space vector PWM for voltage source inverters using artificial neural networks based on FPGA", Proceedings of the International Forum of Strategic Technology (IFOST) 2010, pp 1-6, 2010

[7] E. Monmasson, L. Idkhajine, I. Lahoucine, M. N. Cirstea, I. Bahri, A. Tisan and M. W. Naouar, "FPGAs in industrial control applications", IEEE Transactions on Industrial Informatics, vol. 7, no. 2, pp. 224-243, 2011

[8] R. Salomon and R. Joost, "BOUNCE: A new high-resolution time-interval measurement architecture", IEEE Embedded System Letters (ESL), vol. 1, no. 2, pp 56-59, 2009

[9] E. Heinrich, M. Lüder, R. Joost and R. Salomon, „ X-ORCA – a biologically inspired low-cost localization system", Proceedings of the 10th International Conference on Adaptive and Natural Computing Algorithms, pp. 373-382, 2011

[10] J. Kalisz, "Review of methods for time interval measurements with picosecond resolution", Metrologia, vol. 41, no. 1, pp. 17-32, 2004

[11] F. Stüpmann, „Self-learning neural structur – an approach towards analog hardware implementations of neural networks", original title: "Selbständig lernende neuronale Struktur – ein Beitrag zur analogen Hardwarerealisieurng neuronaler Netze", PhD-Thesis, University of Rostock, 2001

[12] P. Sedcole and P. Cheung, "Within-die delay variability in 90nm FPGAs and beyond", Proceedings of the International Conference on Field Programmable Technology 2006, pp. 97-104, 2006

[13] Altera Corp., "Guaranteeing Silicon Performance with FPGA timing models", whitepaper, 2010

[14] R. Rojas, "Neural networks – a systematic introduction", Springer, New York, 1996

[2] The numbers of resources from the product data sheets were calculated to four-bit input look-up tables equivalents to ensure comparability. See manufactures websites for more information.

[15] A. Thompson, "Notes on design through artificial evolution: opportunities and algorithms", Proceedings of the 5th International Conference on Adaptive Computing in Design and Manufacture (ADCM), pp 17-26, 2002

[16] J. Mason, P. S. Linsay, J. J. Collins and L. Glass, "Evolving complex dynamics in electronic models of genetic networks", Chaos, vol. 14, no. 3, pp. 707-715, 2004

[17] Y. Zhang, P. Huang and R. Zhu, "Upgrading of integration of time to digit converter on a single FPGA", Proceedings of the 15th International Laser Ranging Workshop, October 2006