Emulation of Artificial Neural Network on an FPGA-based Accelerator using CYCLONE II

Sarita Chauhan, Bahadur Singh, Bhajanlal Vishnoi, Subhash Saini, Vikas Kala

Abstract:-Analog VLSI circuits are being used successfully to implement Artificial Neural Networks (ANNs). These analog circuits exhibit nonlinear transfer function characteristics and suffer from device mismatches, degrading network performance. Because of the high cost involved with analog VLSI production, it is beneficial to predict implementation performance during design. We used hardware timemultiplexing to scale network size and maximize hardware usage. An on-chip CPU controls the data flow through various memory systems to allow for large test sequences.We show that Block-RAM availability is the main implementation bottleneck and that a trade-off arises between emulation speed and hardware resources. However, we can emulate large amounts of synapses on an FPGA with limited resources. We have obtained a speedup of 30.5 times with respect to an optimized software implementation on a desktop computer.

Keywords:-Artificial neural networks, analog VLSI emulation, FPGA-based accelerators, hardware time multiplexing, embedded systems.

I. INTRODUCTION

Artificial neural networks (ANNs) have learning capabilities that are used in a variety of applications, such as face recognition, motor control, automated medical diagnosis, signal decoding and data mining. ANNs simulate biological neural networks in order to model complex relations between inputs and outputs of a network . According to the perceptron neuron model, ANNs consist of simple processing elements called artificial neurons, which in turn consist of artificial synapses. Mathematically, an artificial synapse multiplies a stored weight value with an input value. To use ANNs practically, an adaptive algorithm changes the weight values contained in artificial synapses so that the network output converges over time to a desired value. The desired value can be given to the network as an input (supervised learning) or the algorithm can determine these desired values for itself (non-supervised learning). This convergence of weight values represents the previously mentioned ANN capability to learn.

Various implementation methods for ANNs exist today . CPU implementations are an option, but implementations on platforms that allow for parallel processing of data are more efficient due to the parallel nature of ANNs. Furthermore, the computational-intensive nature of ANNs and their algorithms implies that even custom digital Application-Specific Integrated Circuits (ASICs) solutions become constrained by power and size limitations.

Manuscript Received on March 2015.

Smt. Sarita Chauhan, Manikya Lal Verma Textile and Engineering College, Pratap Nagar, Bhilwara, Rajasthan 311001

Bahadur Singh, Manikya Lal Verma Textile and Engineering College, Pratap Nagar, Bhilwara, Rajasthan 311001

Bhajanlal Vishnoi, Manikya Lal Verma Textile and Engineering College, Pratap Nagar, Bhilwara, Rajasthan 311001

Subhash Saini, Manikya Lal Verma Textile and Engineering College, Pratap Nagar, Bhilwara, Rajasthan 311001

Vikas Kala, Manikya Lal Verma Textile and Engineering College, Pratap Nagar, Bhilwara, Rajasthan 311001

Mixed-signal Very-Large-Scale Integration (VLSI) circuits have shown to be a feasible way of implementing ANNs . The problem with the implementation of ANNs in mixedsignal VLSI is that the analog circuits used for the implementation of the neural network suffer nonlinearities in their current-voltage transfer characteristics due to Process/Voltage/Temperature (PVT) spread (device mismatch) and the network learning performance suffers from these variations. We previously compensated for these problems at the cost of chip area , which is not always possible. Since design and production of analog VLSI circuits has high costs and performance is degraded by these nonlinearities, it is beneficial to predict implementation performance during design. A performance prediction tool (emulator) is thus required to foresee the influence of nonlinearities and device mismatch when implementing networks on analog VLSI.

II. BACKGROUND

Our emulator emulates ANNs implemented in analog VLSI. As already noted, we focus on single-neuron ANNs. Furthermore, we focus on the implementation of the Least Mean Squared (LMS) algorithm as a proof of concept. Before exploring the implementation of the emulator, we present brief mathematical concepts for a single neuron in Subsection II-A. Also, we briefly present the transfer functions of the analog VLSI circuits we used to implement artificial synapses and emulation techniques for these transfer functions in Subsection II-B.

II. a Spiking Model by Izhikevich (2003)

All of the responses in Fig. 1 were obtained using a simple model of spiking neurons proposed recently by Izhikevich.

$$v' = 0.04v^2 + 5 * v + 140 - u + I \quad (1)$$

$$u' = a(b * v - u) \quad (2)$$

$$if v >= 30mV then v < -c and u < -u + d \quad (3)$$

Here variable represents the membrane potential of the neuron and represents a membrane recovery variable, which accounts for the activation of ionic currents and inactivation of Na ionic currents, and it provides negative feedback to . After the spike reaches its apex mV , the membrane voltage and the recovery variable are reset according to the (3). If skips over 30, then it rst is reset to 30, and then to so that all spikes have equal magnitudes. The part is chosen so that has mV scale and the time has ms scale. Geometrical derivation of the model based on fast and slow nullclines can be found. The model can exhibit ring patterns of all known types of cortical neurons with the choice of parameters a,b,c,d.

It takes only 13 oating point oper- ations to simulate 1 ms of the model, so it is quite ef- cient in large-scale simulations of cortical networks. We stress that mV in (3) is not a threshold, but the peak of the spike. The threshold value of the model neuron isbetween 70 and-50, andit is dynamic, as in biological neurons. Tobuild intuition and understanding of the dynamics of the model, the reader is advised to download an interactive MATLAB tutorial program from the authors webpage and play with the model and its parameters. In particular, the reader could explore all 20 neuro-computational properties in Fig.

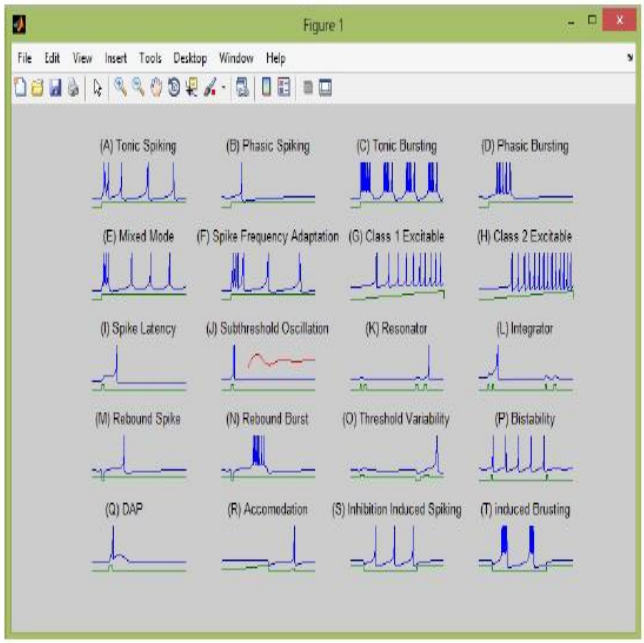

Figure 1: Various Properties of Artificial neural network

II. b Emulation of analog VLSI hardware

An artificial synapse in analog VLSI consists of a multiplier circuit and a memory circuit to store the weight value of a synapse. In previous work, Figueroa et al. produced 64 synapses in a 0.35m CMOS process , using a digital implementation of the LMS algorithm and Pulse-Width Modulator (PWM) . A Gilbert cell topology is used as a current multiplier . A floating-gate pFET transistor is used to implement analog memory in CMOS . See Figure 1 for the nonlinear transfer functions in the produced analog circuits. In the following paragraphs, the emulation of the analog multiplier and memory cells will be detailed.1) Multiplier cell: A Gilbert cell multiplier has a transfer function as given in . Here, Iout is the output current, I0 is the multiplier cell saturation current, UT is the thermal voltage and Vw,x are the inputs of the multiplier. Figure 1(a) shows nonlinearities for eight analog multipliers for varying weight voltages, keeping the input voltage constant.

$$I_{out}(t) = I_0 \cdot \tanh \frac{V_w(t)}{2U_T} \cdot \tanh \frac{V_x(t)}{2U_T}$$

In order to emulate an analog multiplier cell, we measured the transfer functions of the Gilbert cells by increasing Vw with constant step size. We then fitted these measurements with tanh-curves (4). Here, the sampled versions of Iout(t),Vw(t),Vx(t) are denoted as yk,wk,xk, respectively. Furthermore, Aw,k, Bw,k and Cw,k are the fitting parameters which represent the analog VLSI transfer functions. Finally, the tanh is approximated with the first order Taylor approximation, which is sufficiently precise for small xk. Xx,k and Yx,k are the first order Taylor approximation parameters

$$y_k(w_k, x_k) \approx A_{w,k}.tanh(B_{w,k}.x_k) + C_{w,k} \qquad (4)$$

Memory cell:-The output voltage Vw of an analog VLSI memory cell depends linearly on the amount of electrons stored on the oating gate as can be seen from Figure1 (b). However, each memory cell transfer function has a di_erent slope due to PVT spread [11]. The amount of weight change per electron pulse is denoted slopeTF. The amount of electron pulses (denoted Npulses,k), needed for the required voltage change wk, is calculated by the algorithm using (5). The memory cell weight voltage is changed by adding the pulses to the oating gate using the digitally implemented PWM.

$$N_{pulses,k} = \frac{\Delta w_{real,k}}{slope_{TF}}$$

To emulate the memory cell, we change the stored weight value relative to the change in the number of pulses in the analog implementation. Mathematically, each memory cell changes the weight value wk according to the weight change wk calculated by the algorithm. In , the required weight change is then given by wreal,k. The approximated version of this weight change is denoted wapprox,k. The remainder Rk represents the deference between the re- quired weight change and the calculated weight change, which arises since anite amount of steps are used to represent the linear transfer function.

$$\Delta w_{real,k} = \Delta w_k + R_{k-1} \qquad (5)$$

$$\Delta w_{approx,k} = N_{pulses,k}.slope_{TF} \qquad (6)$$

$$R_k = \Delta w_{real.k} - \Delta w_{approx,k}$$

III. Implementation

In this section we show how the mathematical descriptions from the previous section are mapped to an implementation on cyclone II DE2 kit. First, we explain our proposed hardware re-use technique in Subsection III-A. Secondly, we describe system data flow and hardware/software interaction in Subsection III-B. Finally, we give an overview of the emulator hardware and describe the hardware implementation in Subsection III-C.

III. a Temporal synapse slicing

As noted in the introduction, we propose to solve resource constraints through the re-use of hardware blocks to enable emulation large-size networks. We will from now on refer to this technique as temporal synapse slicing. A temporally sliced synapse consists of an emulated multiplier cell, an emulated memory cell and control hardware, which together emulate the function of one artificial synapse as implemented on analog VLSI. We re-use this single temporally sliced synapse, physically implemented on the FPGA, to emulate multiple artificial synapses over time.

A temporal slice refers to all temporally sliced synapses in the system together at a single point in time. For example, the first slice are all physical synapses operating to emulate the first artificial synapse they represent. A combination of a number of slices and a number of temporally sliced synapses creates a neuron. For example, when we operate 5 slices with 5 physically implemented temporally sliced synapses, we emulate a 25-synapse neuron. See figure 2. For each sample, all sliced synapses are operated sequentially: one synapse is operated M K times. Figure 3 shows (schematically) an example of the neuron structure as it arises through the use of the synapse slicing. In this example we show M slices and 2 temporally sliced synapses per slice to form a 2M-synapse neuron. Each synapse contains a memory cell emulator, a multiplier cell emulator and a slice adder. The synapse outputs y1,k and y2,k are intermediate slice results, added by a third adder to form the total network output yk. Furthermore, the input data block shown in the figure contains a set of K input samples for each network input. One sample is fed to the network inputs in parallel and samples follow each other sequentially, until the whole data block has been processed. Finally, the algorithm processes network information to calculate weight updates, using the LMS algorithm as given.

Note that, to explain the principle, only 2 temporally sliced synapses are shown. In practice, as much hardware synapses would be implemented on an FPGA as possible, with the amount of slices required to scale the network to the required size. The amount of synapses contained within the single-neuron ANN is the multiplication of the amount of slices (M) with the amount of temporally

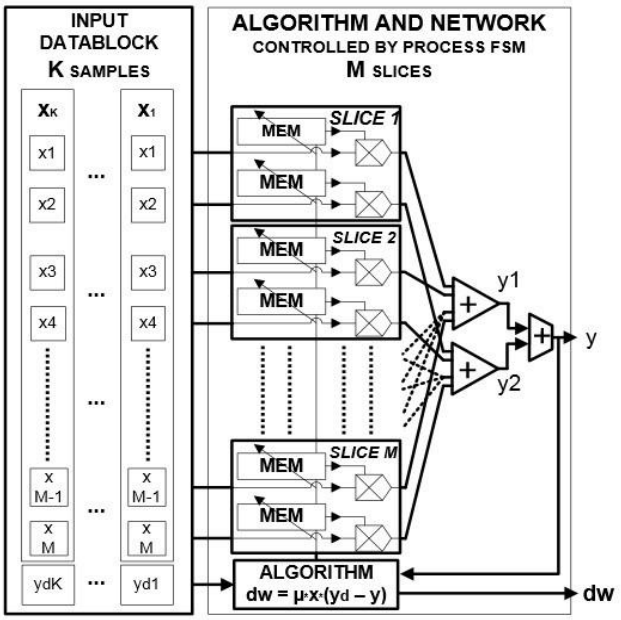

Figure 2: An example neuron topology using sliced synapses. One temporally sliced synapse consists of one multiplier cell and one memory cell. One slice contains two synapses. One adder per synapse tracks intermediate slice outputs.

Sliced synapses. In theory, this allows for the emulation of thousands of artificial synapses. However, since execution time increases linearly when more slices are used, a practical limit exists and a trade-off arises between hardware usage, time usage and network size.

III. b Data flow and HW/SW interaction

To allow for large tests (10k samples) to be performed with the emulator, sufficiently large memory systems are required. We used a DE2 kit for the implementation of the emulator system .The following hardware is provided on the DE2 board Altera Cyclone II 2C35 FPGA device, Altera Serial Configuration device - EPCS16 , USB Blaster (on board) for programming and user API control, both JTAG and Active Serial (AS) programming modes are supported ,512-Kbyte SRAM ,8-Mbyte SDRAM, 4-Mbyte Flash memory (1 Mbyte on some boards), SD Card socket, 4 pushbutton switches,18 toggle switches, 18 red user LEDs, 9 green user LEDs, 50-MHz oscillator and 27-MHz oscillator for clock sources, 24-bit CD-quality audio CODEC with line-in, line-out, and microphone-in jacks, VGA DAC (10-bit high-speed triple DACs) with VGA-out connector, TV Decoder (NTSC/PAL) and TV-in connector, 10/100 Ethernet Controller with a connector, USB Host/Slave Controller with USB type A and type B connectors, RS-232 transceiver and 9pin connector, PS/2 mouse/keyboard connector, IrDA transceiver, Two 40-pin Expansion Headers with diode protection.

Figure shows the data flow as we have implemented it

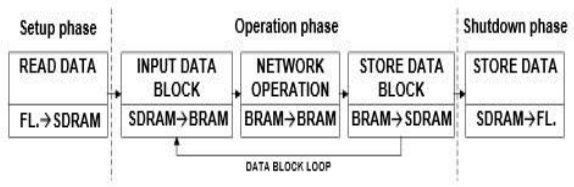

Figure 3: The data flow, which is operated by the CPU and implemented in software, is separated in phases (setup, operation and shutdown). During each phase, data is copied from one source to another, sample-per-sample or block-per-block.

On the CPU. First, the PPC handles samples one-by-one, copying them from the flash memory (denoted FL. in the figure) and storing them on SDRAM. Samples are divided in data blocks, which are processed by emulator hardware before the output is finally written to the flash memory. Data stored on flash memory is not divided into smaller blocks which fit SDRAM, first due to simplicity and second because a 512MB SDRAM can already store around 134 million 32-bit samples, which is sufficient for most development or research applications. We thus use the flash system only to provide a practical means to copy data from and to the FPGA.

The PPC is required to interact with the emulator hardware and monitor progress until operation is finished. We used a handshaking protocol using Software-Accessible Registers (SARs) to implement hardware/software interaction. SARs are reserved registers with read/write accessibility for both the CPU and the FPGA hardware. The term SAR has been coined by Xilinx. We will from now on refer to the handshaking protocol as polling. It operates as follows. First, the custom hardware is enabled by the PPC through a SAR.

Then the processor continuously reads out and compares the value contained within a second SAR, while the hardware is operating. Finally, the emulator hardware is disabled by the CPU when the hardware sets a ready flag. There are some disadvantages to the polling approach. One disadvantage is that the processor cannot perform other tasks while waiting for the emulator hardware to finish. A second disadvantage is that the PPC wastes power while the emulator hardware operates. However, no other tasks but data management are required from the processor in this implementation. Advantages of the polling protocol include small resource requirements, omittance of a set-up time and ease of implementation. Furthermore, the V2P includes a second PPC processor to perform other tasks if it would be required.

III. c Emulator hardware design

The emulator hardware is shown schematically in Figure 5. First, the process controller is shown, which is a Finite State Machine (FSM) that starts/monitors all FSMs within other hardware blocks and controls two counters.Secondly, the Synapses block contains all temporally sliced synapses. Thirdly, the Algorithm block consist of a number of pipelined hardware multipliers, calculating all weight update values. Finally, the Addition block is simply an adder for synapse outputs. In subsequent parts of this section, hardware designs for each block will be detailed.

Process controller: The process controller is the main FSM for the hardware system. This controller implements the slicing system. The FSM diagram is shown schematically in Figure. The FSM starting state is denoted Idle and the FSM ending state is denoted Datablock ready. The flags involved between states are indicated in the figure. For example, in state Await Alg, the ready signal is the ready flag from the algorithm, signaling the process controller it is ready operating.

First, the CPU enables the FSM through a SAR. Then, the multipliers retrieve the value stored in the memory and multiply it with the first input sample for all slices (denoted slice loop 1). Then, outputs w and y are stored in an output BRAM. The algorithm is started to calculate the weight updates. When ready, the memory cells are started, updating their weights to contain the new weight value calculated by the algorithm (denoted slice loop 2). The system executes the described process for all samples contained in the data block (denoted sample loop), after which a ready flag is set in a SAR, notifying the CPU that the hardware is ready for the next data block. Throughout this process, the sample and slice index values are updated to allow correct data selection from the input data BRAM and control of the amount of loops in the process FSM. The relative simplicity of the process controller FSM shows that our proposed temporal slicing technique can be used in practical systems. By increasing the amount of slice loops, we can linearly scale up the neuron size.

Temporally sliced synapse: The hardware block emulating the artificial synapse consists a multiplier cell emulator and a memory cell emulator. Also, it contains a slice BRAM. The slice BRAM contains for each slice in the synapse

- the memory cell BRAM index nk1 pointing to the current weight;
- the w for the next sample;
- the remainder value Rk arising from weight approximation.

Multiplier cell: The multiplier cell is used to calculate the yk value for a synapse. The hardware consists of two BRAMs, a single pipelined hardware multiplier and an FSM controller.

The first BRAM contains Aw,k, Bw,k and Cw,k for N measurements for one analog multiplier cell transfer function. Each multiplier cell emulates one of the measured

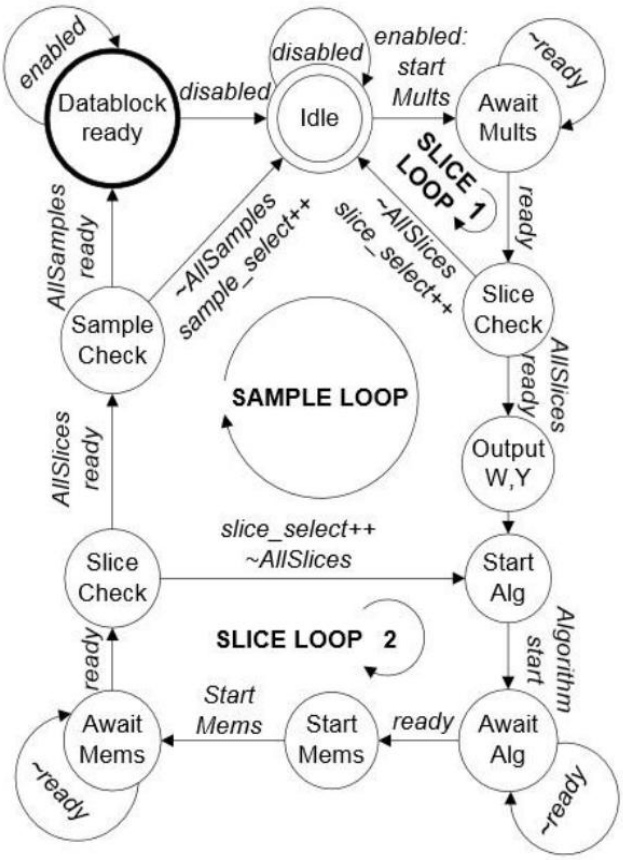

Figure 4: Process controller FSM diagram (schematic). The double lined state is the starting state, the bold lined state the ending state.

analogy VLSI multiplier cells. A disadvantage to this way of implementing the slicing system is that it does not account well for PVT spread if the amount of slices is much higher then the amount of temporally sliced synapses. Therefore, it is essential that as much temporally sliced synapses as possible are implemented. The second BRAM contains Xx,k and Yx,k for M approximation points, so that a higher M gives a higher resolution of the tanh approximation.

The multiplier FSM operates as follows. The multiplying process starts with extraction of the Aw,k, Bw,k and Cw,k parameters. Second, we extract the Xx,k and Yx,k parameters from BRAM. In subsequent steps, we calculate the output using the extracted parameters. Because the tanh we approximate is an odd, inversely symmetrical function, we calculate the absolute value of Xx,k*B*xi+Yx,k result to obtain double resolution with the same memory space. Memory cell:

The memory cell design consists of a Look-Up Table (LUT) for weight values, a divider and time changing the weight value to the data saved in slice memory. We assume that the learning rate, , is constant and we simplify it to be a power of 2. We implemented it as a bit shift. The algorithm block consists of multipliers, into which we feed the input data sequentially, and an FSM controller. We can set the amount of multipliers prior to synthesis, depending on the amount of synapses that are required. A minimum of one hardware multiplier is required. If the network consists of a large amount of synapse blocks, more hardware multipliers can be used, so we can linearly exchange hardware for algorithm execution speed.

IV. RESULT

In this section, we present our simulation results, postimplementation timing results and post-implementation resource results for the analog VLSI ANN emulator. We show that a trade-off between time and hardware resources arises.

1. First we write Verilog in Quartus of Neuron synapses and compile it.
2. After compilation , we select functional simulator tool and give the input to each node , constant (a,b,b,c,d) , membrane potential of the neuron(v) and represents a membrane recovery (u) and the duty cycle to 50
3. After it we generate Functional Simulation Netlist and get output waveform on Quartus.
4. We burn program on DE2 kit
5. After burning program we get output on CRO.

1). The basic system:-The first system consists just of the custom emulation hardware and BRAM to store test data. It does not require any inter operation with the CPU or external memory systems. Slicing is not possible and there is one data block which stores all samples on-chip. Data is read out from the system through a
serial link;

2). The slicing system:-The second system is the system as described in this paper. It operates with temporally sliced synapses and feeds numerous data blocks into the system as described in Subsection III, storing both input and resulting output data on the flash memory system as described in Subsection III-B. The data is extracted from the flash memory.

A. Simulation: emulation of a small test network We simulated the systems as follows. First, we generated a random input data set of 1024 samples, which is small enough to fit also on the basic system, where only BRAM is available to store the input data. Then, using the generated data set, we produced a reference output signal (dk) by fixing the weights of a 5-synapse neuron. For both systems, we then implemented 5 synapses in the emulator using 5 (randomly selected) measurement data sets from the total set of 64 analog circuits. Finally, we then simulated both systems using the previously generated inputs in the Xilinx ISE Simulator. Note that we configured the slicing system to use 1 slice and (the same selection of) 5 hardware synapses to mirror the basic system.

The results produced by both systems are identical. Figure 7 shows weight evolution for the emulated 5-synapse neuron. The weight values converge towards the same fixed values we set when generating the reference data. Furthermore, the simulation results were verified to be identical with our previous CPU implementations of the emulator. Next to convergence of the weights for a single slice, we also simulated the process controller to verified its operation, also for multiple slices. Multipliers first operate once for all slices. Then, for each slice subsequently, first the algorithm and then the memory cell are operated. This operation order is in accordance to the intended operation as previously.

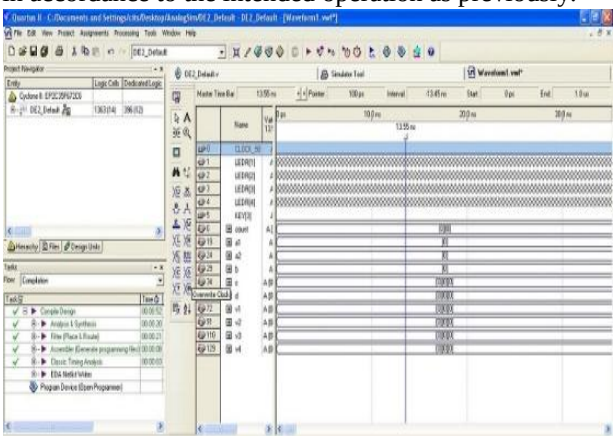

Figure 5: For input we select the value of a,b,c,d , current I, potenial v and u

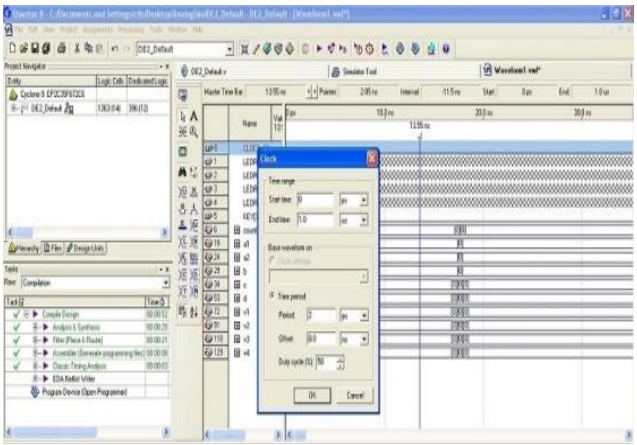

Figure 6: Set the clock

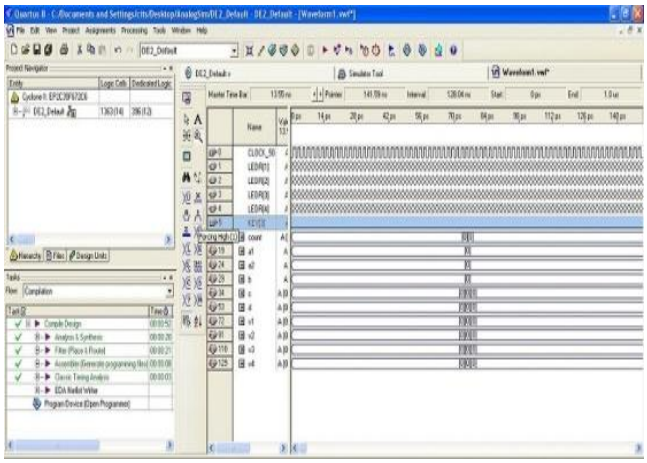

Figure 7: Set the clock and high input for led switch

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

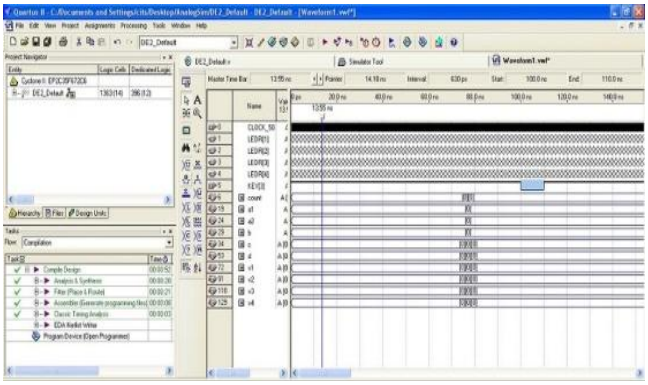

Figure 8: Set the clock and high input for led switch

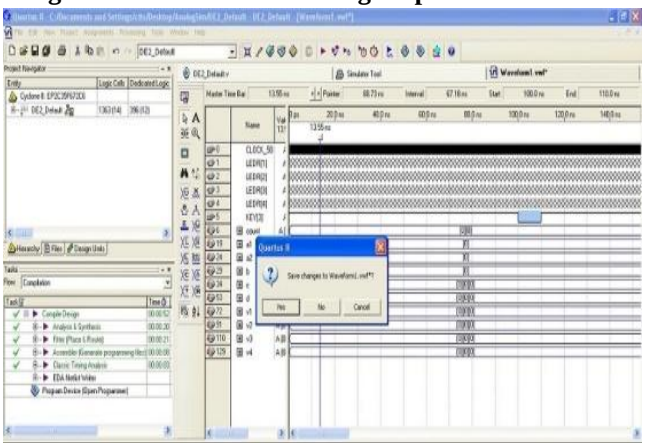

Figure 9: save the changes done in wavefrom

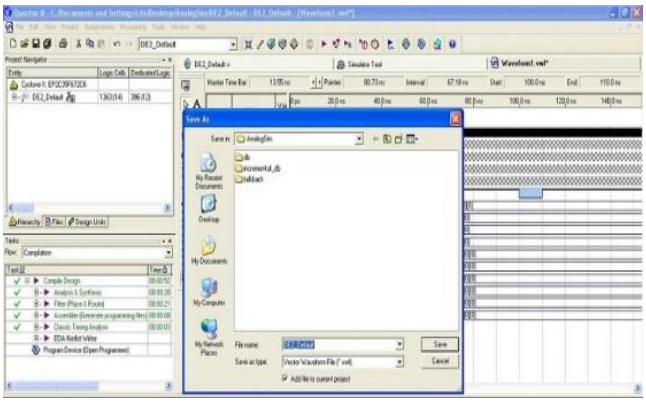

Figure 10: save the changes done in wavefrom

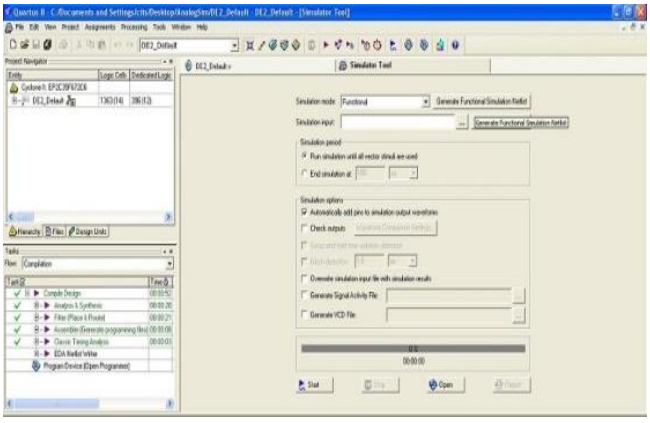

Figure 11: Generate the netlist function generator

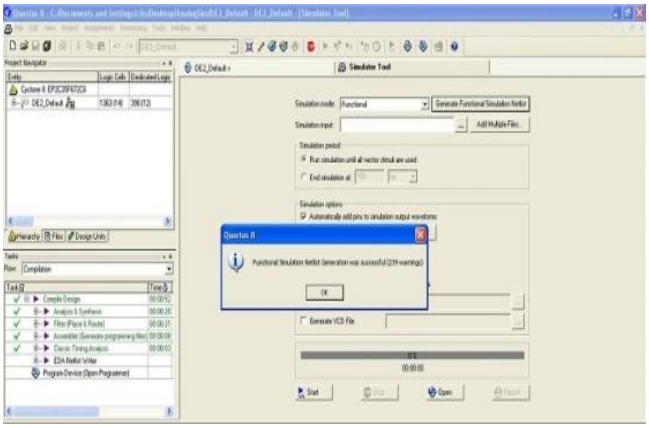

Figure 12: simulate the netlist function generator

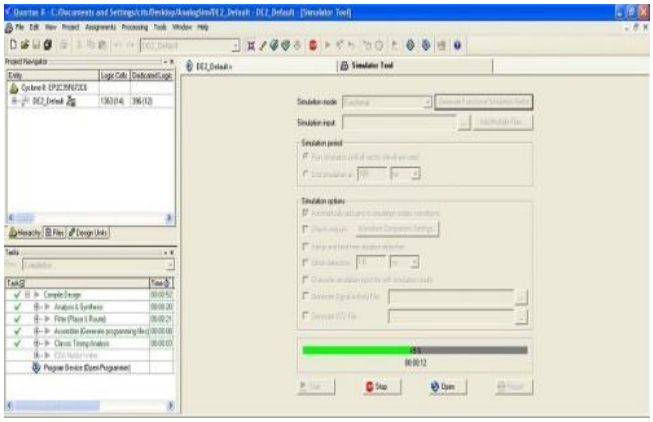

Figure 13: sucessful simulation window

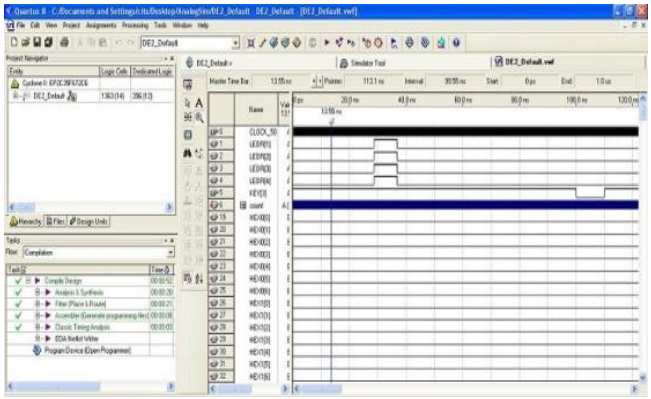

Figure 14: Count waveform of neuron

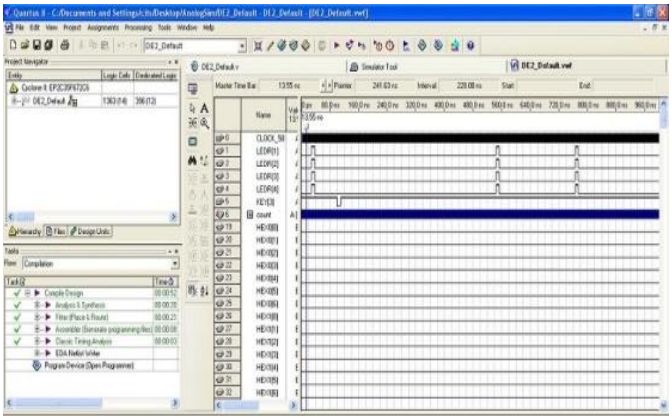

Figure 15: Count waveform of neuron

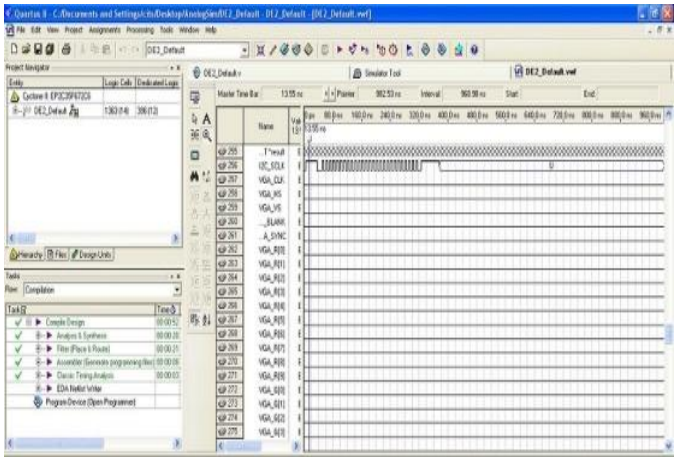

Figure 16: Count waveform of neuron

Figure 17: Spiking of neurons when input is (0 0)

Figure 18: Spiking of neurons when input is (0 1)

Figure 19: Spiking of neurons when input is (1 0)

B. Required hardware resources

For hardware we use DE2 kit that provide FPGA based

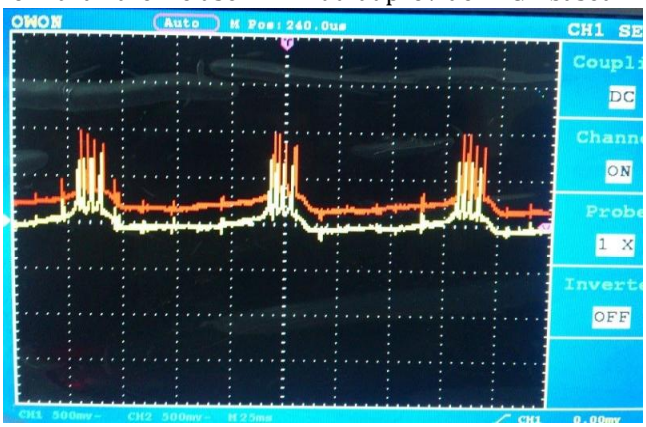

Figure 20: Spiking of neurons when input is (1 1)

platform. FPGA based platfrom provide us to capability to simulate artificial neural network. FPGA based platform have parallism property that have artificial neural network.

C. Timing evaluation

After the Place And Route (PAR) process, we analyzed the system timing details for a single-neuron analog VLSI emulator with 2 slices on the DE2 kit. Where possible, we measured in-operation timings to verify the simulated values. All post-implementation system delays are simulated to be ¡ 10ns, such that the network can operate using the DE2 kit maximum clock of 100MHz. When we implement the algorithm using 5 hardware multipliers (one per synapse), the hardware requires 16, 14 and 7 cycles per sample for respectively the multiplier, memory and algorithm blocks. In total, the process controller requires 100 cycles/sample, which results in a processing speed of 1MSps (Samples per second) for the 5-synapse, 2slice slicing system (CPU cycles required to load data blocks from flash and SDRAM memory systems to the hardware not included). An average 50 cycles/slice/sample is thus required.

To the best of our knowledge, our work publishes the first accelerator for this application. As such, we were not able not make state of the art comparisons. However, in comparison to our previous CPU implementations, a significant speed-up has been obtained. We compared the operation times (in cycles/sample) for our FPGA accelerator (5 synapses, 1 slice, 100MHz clock speed) with both a Matlab- and C-implementation of the emulator (5 synapses, 2GHz clock speed). To this end we did the same 1024sample test on all implementations. Our emulator (0.5 s/sample) shows a speed-up of 5400 (2700 s/sample) and 30.5 (15.1 s/sample) in comparison to the Matlaband C-implementation, respectively. It is expected that a V6 FPGA would perform 2-3 magnitudes better than the optimized C-implementation due to the higher clock speed and the higher maximum number of artificial synapses that can be implemented.

We conclude that temporal synapse slicing allows for a trade-off between time and hardware resources, which was previously not available.

To visualize this, we show a number of possible configurations when 100 artificial synapses need to be implemented and their costs in terms of time and hardware resources in Table III.

V. CONCLUSIONS

The contributions of this work are twofold. We implemented an FPGA-based accelerator for practical emulation of analog VLSI neural networks and investigated the limits that availability of FPGA resources impose on the amount of synapses that we can emulate. First, we conclude that emulation of large analog VLSI neural networks is feasible on an FPGA platform. Secondly, we conclude that availability of on-chip memory limits the amount of test samples, but external memory systems overcome this limitation. Our emulator allows for emulation of nonlinearities of analog VLSI implementations for artificial neural networks. The emulator enables convergence and performance analysis of large single-neuron ANNs. We show that it is possible to implement 500+ synapses even on an entry-level FPGA with limited resources. We use hardware efficiently through temporally slicing of synapse emulator blocks and show there is a trade-off between resources and emulation speed. Furthermore, we show that external memory systems and a CPU for data flow control together overcome the limitations posed by available on-chip memory regarding the amount of input samples, allowing for test sequences of more then 10K samples. Finally, our Virtex 2 Pro accelerator obtains a speedup of the order of one magnitude compared to a specialized software implementation, while it is expected that a similar implementation on a state of the art FPGA such as the Virtex 6 could obtain a speedup of 2-3 magnitudes.

VI. FUTURE APPLICATION

Future work aims at emulation of multiple layer/neuron networks and the use of more complex algorithms such as Independent Component Analysis (ICA). Also, the current implementation is not user friendly and the user requires knowledge of the inner workings to modify the architecture. For future work, we want to create a user friendly emulator tool which can be used by designers of mixed-signal VLSI and researchers on ANNs in the field. This includes tools for data generation, implementation and network analysis. We will work to enable implementation of different algorithms and circuits by changing equations and measurement data, respectively.

REFERENCES

1. C. M. Bishop, Pattern recognition and machine learning. Springer Science+Business Media, LLC, 2006.
2. C. Diorio, D. Hsu, and M. Figueroa, Adaptive CMOS: from Biological Inspiration to Systems-on-a-Chip, Proceedings of the IEEE, vol. 90, no. 3, pp. 345357, 2002.
3. G. Cauwenberghs and M. A. Bayoumi, Eds., Learning on Silicon: Adaptive VLSI Neural Systems, ser. The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Press, 1999.
4. M. Figueroa, S. Bridges, and C. Diorio, On-chip compensation of device-mismatch effects in analog VLSI neural networks, in Advances in Neural Information Processing Systems 17. Cambridge, MA: MIT Press, 2005.
5. B. Dolenko and H. Card, Tolerance to Analog Hardware of On-Chip Learning in Backpropagation Networks, IEEE Transactions on Neural Networks, vol. 6, no. 5, pp. 1045 1052, 1995.
6. E. Matamala, Simulation of adaptive signal processing algorithms in VLSI (in Spanish). Civil Electrical Engineers thesis, Universidad de Concepcion, 2006.
7. D. B. Thomas, L. Howes, and W. Luk, A Comparison of CPUs, GPUs, FPGAs, and Massively Parallel Processor Arrays for Random Number Generation, in Proceedings of the ACM/SIGDA international symposium on FPGAs, 2009, pp. 6372.
8. D. Herrera and M. Figueroa, FPGA-based Analog VLSI Neural Network Emulator, in Proceedings of the Chilean Congress on Computing, 2008.
9. F. Yang and M. Paindavoine, Implementation of an RBF Neural Network on Embedded Systems: RealTime Face Tracking and Identity Verification, in IEEE Transactions On Neural Networks, vol. 14, 2003, pp. 11621175.
10. V. Stopjakov, D. Miuk, L. Benuskova, and M. Margala, Neural Networks-Based Parametric Testing of Analog IC, in IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, vol. 17, 2002.
11. M. Figueroa, E. Matamala, G. Carvajal, and S. Bridges, Adaptive Signal Processing in Mixed-Signal VLSI with Anti-Hebbian Learning, in IEEE Computer Society Annual Symposium on VLSI. Karlsruhe, Germany: IEEE, 2006, pp. 133138.
12. D. Coue and G. Wilson, A four-quadrant subthreshold mode multiplier for analog neural-network applications, Neural Networks, IEEE Transactions on, vol. 7, no. 5, pp. 1212 1219, sep 1996.
13. C. R. Schneider, Analog CMOS Circuits for Artificial Neural Networks, Ph.D. dissertation, University of Manitoba, 1991.
14. C. Diorio, S. Mahajan, P. Hasler, B. A. Minch, and C. Mead, A High-Resolution Nonvolatile Analog Memory Cell, in IEEE International Symposium on Circuits and Systems, vol. 3, Seattle, WA, 1995, pp. 22332236.
15. S. Kilts, Advanced FPGA Design, Architecture, Implementation and Optimization. Wiley-Interscience, 2007.