A SELF-ORGANIZING LEARNING ARRAY AND ITS HARDWARE-SOFTWARE CO-SIMULATION

Janusz Starzyk* and Yongtao Guo*

Abstract **– In this paper, a self-organizing learning array (SOLAR) and its hardware-software (HW/SW) co-simulation are presented. In SOLAR, every neuron maximizes its information index during feed forward self-organizing learning. SOLAR was simulated on benchmark examples and showed good ability to learn exceeding the performance of many traditional classifier algorithms. A simple HW/SW co-simulation method was adopted to avoid the use of two simulators and complex inter-process communication. In this co-simulation method, software is modelled using behavioral, and hardware using structural hardware description language (VHDL) models. Their interface is also modelled in VHDL.**

1 INTRODUCTION

Artificial Neural Networks (ANNs), derived from the field of neuroscience, display interesting features such as parallelism, adaptation, and ability to learn. Classifier design is an important application of ANNs in such areas as metrology, microbiology, and radar based target recognition. There are many methods to implement classifier including learning machines [1], neural networks [2] and statistical algorithms [3]. SOLAR is a parallel signal processing hardware with relatively sparse interconnections between processing components (neurons). It differs from classical ANNs in the way it is organized and how it learns. Its most important advantage over ANNs is that it scales well in hardware. While classical ANNs are wire dominated (wiring area grows as a cube of the number of neurons), SOLAR's interconnection area grows almost linearly with the number of neurons. SOLAR classifier performed well in comparison with many specialized machine learning algorithms and outperformed all ANNs [4]. Its idea is derived from both neural networks and information theory. SOLAR hardware organization was reported in our pervious work [5]. It will be explored further in this paper.

Considering SOLAR has complex self-organizing architecture and performs very data-intensive computing, we adopted hardware-software co-design approach. In hardware-software co-design, simulation is performed by combining a specific software code (for instance, C++ program, assembly code, or MATLAB routines) with structural hardware models. A hardware description language (like VERILOG or VHDL) is used for hardware modelling and simulation, so typically, at the design of mixed-mode systems (with hardware and software components), a VHDL code is combined with other software for system prototyping and debugging. This is often an error-prone routine requiring filters to handle various formats of data and processed signals. In implementing SOLAR architectures, we developed a hardware-software co-simulation approach to model hardware and software in the same hardware description language program. This approach is simpler than Bassam's co-simulation method [6] since we do not need to consider the software synthesis using their proposed S-graph-based synthesizer, where the software part would be transferred to C language. We focus on the system simulation to verify the critical timing requirement and attain optimal hardware-software participation. In this co-simulation, the software part is implemented using behavioral VHDL description. This part is not synthesizable in FPGA. Structural VHDL is employed to implement neuron's training architecture and this part is synthesizable in FPGA. This co-simulation method simplifies the simulation environment and speeds up SOLAR prototyping.

The rest of this paper is organized as follows. In Section 2, the self-organizing learning array architecture is discussed. Section 3 deals with the fast HW/SW co-simulation using behavioral and structural VHDL description. A summary is given in Section 4.

2 SOLAR ARCHITECTURE

In SOLAR, we adopt feed forward network structure for its stability and fast learning. SOLAR architecuture is divided into three main layers as shown in Fig. 1– input, processing and output layers. The interconnections are randomly initialized at the beginning of training. The basic building blocks of the arcitecture are the small neurons, trained using the entropy-based algorithm [4].

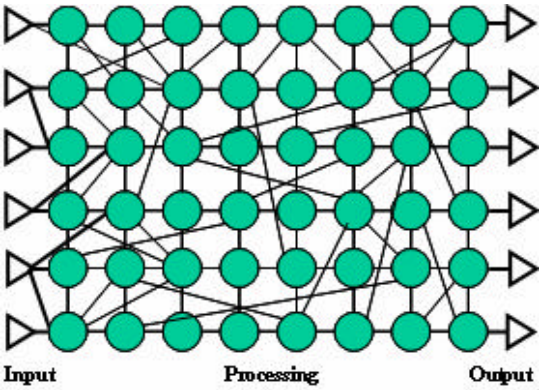

Figure1: Basic SOLAR structure

* School of Electrical Engineering and Computer Science, Ohio University, Athens, OH 4570, U.S.A.
E-mail: [starzyk, gyt]@bobcat.ent.ohiou.edu, tel.: +001740 593 1580, fax: +001740 593 0007.

There are two types of connections to every neuron. The local connections are asscociated with higher connection probabilities, and the remote connections have smaller probilities. In the SOLAR architecture, statistically determined Manhattan distance is used to set the initial connections. In the feed forward structure, the neuron located at a given row and column should always be connected to the one at the same row and previous column. The next nearest neurons are those located at two neurons away from the connecting neuron with certain probability and the remote neurons are randomly selected from all the previous layers including the primary inputs. This interconnection approach applies to both the neuron's input signals as well as the neuron's control signals which define the learning subspace for each neuron.

3 SOLAR CO-SIMULATION USING VHDL

To cope with both the significant NRE cost of custom hardware and the limited programmable hardware resource of FPGAs, the SOLAR implementation is based on tightly coupled conventional processor (software) with configurable logic (hardware) on a single PCI-based VIRTEX XCV800 FPGA card [7]. The software part runs on the host PC. The time consuming part – neuron's self-organizing learning runs in the FPGA chip on the PCB board. The interface between them is via PCI bus. To develop this system, we need real-time test environment to co-verify the HW/SW parts and their interface. This method is time-consuming and it is hard to monitor the interface signal timing.

In this paper, we use the VHDL simulation to explore feasibility of virtual HW/SW prototypes including their interface and then map the resulting SOLAR onto a mixed HW/SW architecture to model the real-time system, and to obtain the operating system characteristics. The existing co-simulation methods contain at least two simulators for both software and hardware respectively integrated through complex inter-process communication. Those methods lack portability and change from simulator to simulator depending on both the hardware and software programming languages. To design SOLAR, we have developed a fast co-simulation method to directly simulate the whole HW/SW system. This method not only facilitates interactive partitioning and complete exploration of the whole system prior to its implementation, but also avoids using two simulators and complex inter-process communication. In the co-simulation environment, we model software using behavioral VHDL description, hardware using structural VHDL building blocks and HW/SW interface using VHDL to describe a finite state machine (FSM) plus input/output FIFOs at RTL level. We can decompose SOLAR co-simulation system into three parts as shown in Fig. 2:

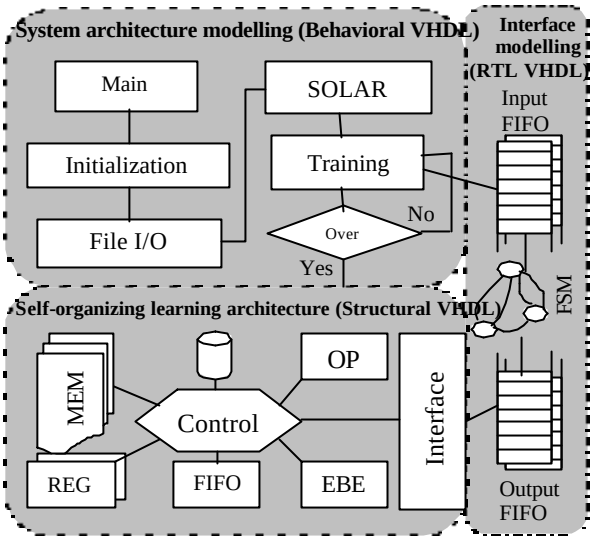

Figure 2: Co-simulation system decomposition

- **System architecture modelling**. This is the software part in the co-simulation using behavioral VHDL description. All of the functional behaviour of the system consisting of many functions is organized in the hierarchical packages as shown in Fig. 3. All functions and signal variables in the packages are shared, and program execution is functionally interleaved. The lower level package is the description for system input and output, initialization and update of the memory element in the network. The initialization is used to build up the 2D neural network mesh architecture with randomly interconnected neurons, set up the threshold update step, and assign initial signal values for system simulation. The higher level packages encapsulate new system functions based on the functions described by the lower level packages. The highest design level function representing the software part in the overall system implements the system organization and management.

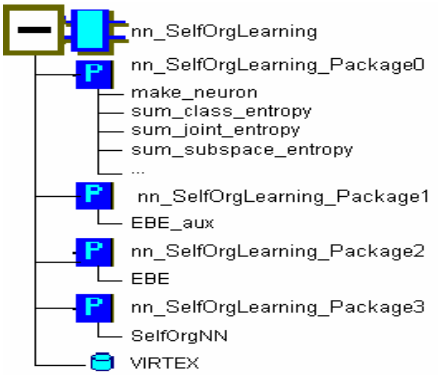

Figure 3: Hierarchy architecture for software model

- **A single neuron's self-organizing learning architecture modelling.** This is the hardware part in the co-simulation using synthesizable structural

VHDL description to model the self-organizing learning process of a single neuron. The neuron's implementation architecture is shown in Fig. 4. Data fed from the interface represents the single neuron's learning subspace. After input data is read into the memory (M) via interface FIFO, the main controller launches optimal threshold searching collaborated by function producing module (OP) and ALU. Finally, the optimal threshold and corresponding information index is stored in several registers (R). These optimal parameters and learning-produced new subspace are read back through rapid DMA transfer requested by the system level functions (software) for storage and further processing.

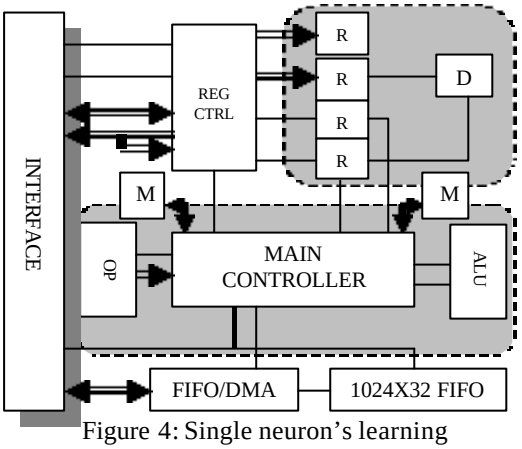

Figure 4: Single neuron's learning

- **Interface modelling.** This is the HW/SW interface part in the co-simulation implemented by FSM and several input/output FIFOs as shown in Fig. 5. This part bridges the gap between the system architecture modelling (software) and the single neuron's self organizing learning architecture (hardware). This part is implemented by a six-state FSM and three FIFOs (six ports) for training, class and other data transfer between hardware and software parts. The FIFO status signals and read/write signals are trigged either by structural VHDL (hardware) or by behavioral VHDL (software) description via FSM to implement the HW/SW communication.

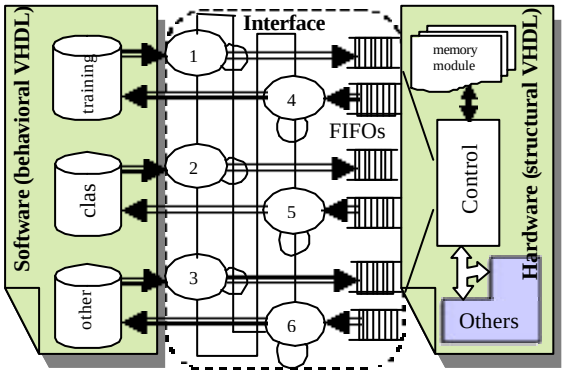

Figure 5: Interface modelling using FSM&FIFO

Co-Simulation Results

We used a single VHDL simulator to simulate SOLAR using the described co-simulation model. As Fig. 6 shows, the HW/SW interface is implemented based on three FIFOs -input, output and address strobe. The communication between software and hardware is controlled by a simple FSM – signal 'SM_STATE' plus those three FIFOs. Some FSM states read the data from software part and send the formatted data to hardware part via input FIFO. Conversely, the other FSM sates read the output FIFO to receive the processed data from the hardware part and send them to software part. The address or data is decided by the address strobe FIFO. This interface based on FSM and FIFO only represents the simplified PCI LOGICORE which is part of a separate firmware connecting PCI bus to FPGA chip. Using co-simulation, we can model functionality essential for system operation leaving out unessential details of the interface protocol.

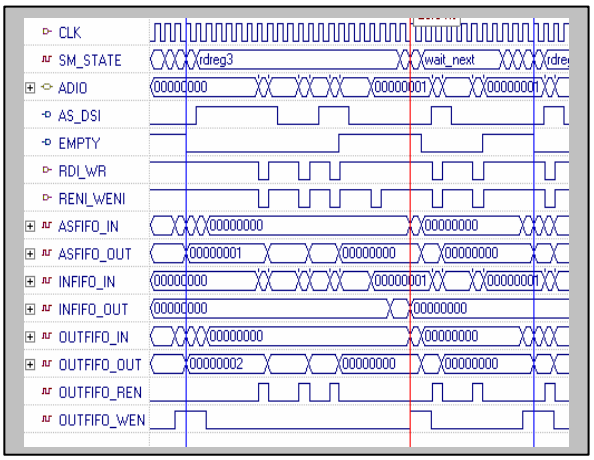

Figure 6: HW/SW interface simulation

The interface process continues until the training process is over as shown in a snapshot simulation waveform in Fig. 7. The training example uses 2 classes in 2-dimensional input space with 1818 training data. Different neurons have different learning subspaces resulting from pseudorandom wiring. Each neuron processes the training data subspace selecting its inputs from the initial pseudorandom selection. It optimises the information index in its learning subspace, setting its threshold, and transformation function. It calculates information deficiency and defines its output subspace threshold clock. This clock is used as an input clock of subsequent neuron, which responds to data from the neuron output space. The illustrated simulation waveform represents only one particular neuron's self organizing learning in a particular learning subspace. The whole system synchronization is controlled by the system modelling part (software). Through the interface, software model sends data to the input FIFOs. The input FIFOs status signals trigger the hardware model to fetch the input

data. After receiving all the data, the hardware module enters into the self-organizing learning stage. During this stage, the hardware module still fetches the control commands from the input FIFO to synchronize the self-organizing learning with software module. Finally, the optimal learned parameters and a new subspace are achieved and the output FIFOs are used as buffers to send the learned results out to software module. In this simulation waveform, the signal "Opt_Threshold" and "ID" represent the optimal threshold and the corresponding information index for this particular training neuron in its learning subspace.

The hardware-software co-simulation results for every neuron correspond to the MATLAB simulation results as far as the final optimal learning parameters except for the difference in data format. Using this co-simulation method, we can promptly verify the hardware-software partition, system functionality, interface efficiency and etc., which saves us much real prototyping time of the entire system.

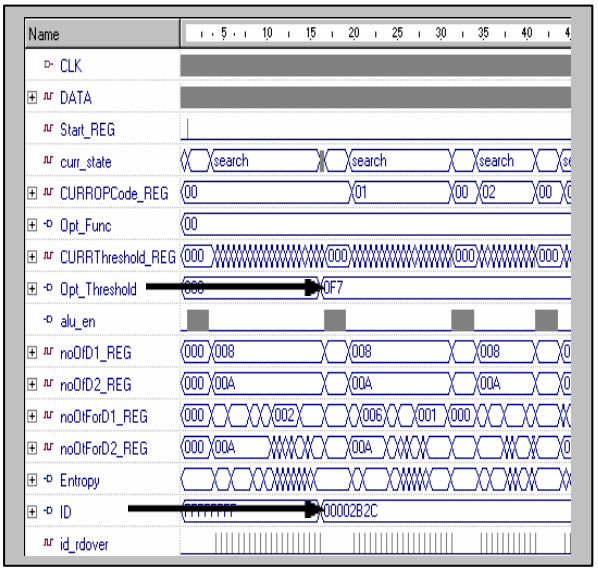

Figure 7: Single neuron's training co-simulation

4 SUMMARY

In contrast to more traditional neural network classifiers or self-organization based on clustering approaches, this paper discusses an information theory based hardware self organization for machine learning. Dynamically reconfigurable architectures [8] can be quickly reconfigured by reading pre-stored configuration bits from memory. SOLAR is similar to these structures, except its reconfiguration is a result of learning, rather than a result of pre-stored architecture. This way it is a new class of learning machines and at the same time a new type of reconfigurable hardware. SOLAR simulation proved it to be advantageous over many traditional learning machines, neural network and statistical methods. Due to the computation-intensive process and large storage requirements, we adopted hardware-software co-design to prototype the system on both microprocessor (software) and a single XILINX VIRTEX XCV800 FPGA (hardware). We present a mechanism for co-simulation of the synthesized hardware and software using a single VHDL simulator. This technique uses behavioral VHDL description to simulate the self-organizing system and to synchronize and control the synthesizable learning neuron's architecture using structural VHDL description. This co-simulation approach achieves fast simulation speed without alternating between two (hardware and software) simulation environments and simplifies hardware and software partition process. Finally, it speeds up our hardware-software co-design to prototype the SOLAR architecture on FPGA.

This design work is to prototype the simplified self-organizing learning array on a single FPGA. Future SOLAR architecture will be based on hundreds of VIRTEX XCV 1000 FPGAs [9] and is currently under design evaluation stage. The presented co-simulation method will benefit the development of such a system.

Reference

[1] Dorigo, M., "Alecsys and the autonomouse: learning to control a real robot by distributed classifier systems," Machine Learning, 19 (3), 209-240, 1995.

[2] R. Anand, G. Mehrotra, C.K. Mohan, and S. Ranka, "Efficient classification for multiclass problems using modular neural networks," IEEE Transactions on Neural Networks, 6:117–124, 1995.

[3] Miller, D., Rao, A., Rose, K., Gersho, A., "A global optimization technique for statistical classifier design," IEEE Transactions on Signal Processing, December 1996.

[4] J. A. Starzyk and Z. Zhu, "Software simulation of a self-organizing learning array system," the 6th IASTED Int. Conf. Artificial Intelligence & Soft Comp (ASC 2002), Canada.

[5] J. A. Starzyk and Y. Guo , "Reconfigurable self-organized nn design using VIRTEX FPGA," the Int. Conf. on Engineering of Reconfigurable Systems and Algorithms (ERSA 2001), Las Vegas, NV, USA.

[6] Bassam Tabbara, Enrica Filippi, Luciano Lavagno, Marco Sgroi, Alberto Sangiovanni-Vincentelli, " Fast hardware-software co-simulation using vhdl models," Design Automation and Test in Europe (DATE), March, 1999.

[7] Nallatech Ltd, "Ballynuey 2 VIRTEX PCI Card Users Guide," 1993-1999.

[8] J. Becker, A. Alsolaim, M. Glesner, and J. Starzyk, "A parallel dynamically reconfigurable architecture for flexible application-tailored hardware/software systems in future mobile communication," the Journal of Super Computing, Erratum Vol. 23, 132, 2002.

[9] Xilinx,"Virtex 2.5V FPGA data sheet," April, 2001.