Exploring Spiking Neural Network on Coarse-Grain Reconfigurable Architectures

Abstract—**This paper presents the architecture and implementation of neural networks on a Coarse-Grain Reconfigurable Architecture (CGRA). In particular, we propose a mapping of Spiking Neural Networks (SNNs) on a CGRA, that guarantees better performance in terms of throughput, latency, area, and power consumption (compared to existing FPGA based implementations). In our design we exploit various CGRA resources to mimic a complex network of neurons and synapses. To test the effectiveness of our approach, we have used Dynamically Reconfigurable Resource Array (DRRA).**

I. Introduction

Today, neural networks becoming an increasingly popular technique to realize learning robotics. Use of neural networks in robotics requires fast and efficient implementation platforms. Recently, the increasing speed and high performance requirements, coupled with the demands for flexibility and low non-recurring engineering costs, have made reconfigurable hardware a very popular implementation technology for neural networks [1]. Today reconfigurable architectures enable partial and dynamic runtime self-reconfiguration [2]. This feature allows the substitution of neural network on a hardware design implemented on this reconfigurable hardware, and therefore, a single device can be adapted to implement various functionalities by simply uploading a new configuration based on neural network application. The reconfigurable architectures can be classified depending on their granularity, i.e., the number of bits which can be explicitly manipulated by the programmer. Coarse-Grained Reconfigurable Architectures (CGRAs) provide operator level configurable functional blocks, word level datapaths, and powerful and very area-efficient datapath routing switches. Compared to fine-grained architectures (like FPGAs) CGRAs enjoy massive reduction of configuration memory and configuration time (two or more orders of magnitude [3]) as well as considerable reduction in routing and placement allocation. All this also results in a significant reduction of the overall area (from 66 % to 99.06 % [4]) and energy consumed per computation (from 88 % to 98 % [4]), though at the cost of a loss in flexibility compared to bit-level operations. Therefore, CGRAs have been a subject of intensive research since the last decade [4].

Most of the existing works that attempt to speed up the neural network algorithm employ FPGAs [5], [6]. They speed up computations by exploiting the parallel processing offered by the FPGAs. However, the FPGAs require fine granular connectivity that is in efficient to handle the complex interconnections required by the neural networks [7]. Compared to FPGAs, network-on-chip (NoC) simplify the connectivity [7] thereby enhance scalability. However, the NoC based solutions (using processor with every node) are inefficient and require complex switches. Therefore, while NoC based solutions provide high scalability, they are unable to provide the parallel processing high speeds offered by the FPGA solutions. In this paper we explore the feasibility of implementing the neural networks on a CGRA. The proposed solution provides the parallel processing of FPGA based implementations with the scalability similar to generic NoCs. Specifically, we consider a phenomenon known as Spike-Timing-Dependent Plasticity (STDP) [8]. To evaluate experimentally the efficiency of the proposed solution on a CGRA, we have chosen the Dynamically Reconfigurable Resource Array (DRRA) [?Shami2012]. Nevertheless, the results obtained in this paper should essentially be applicable to most grid based CGRAs as well.

This paper has two major contributions:

1) We present architecture and implementation to realize a spiked neural networks on a coarse grained reconfigurable array (CGRA);
2) We analyze the scalability and overheads of the proposed technique, on an actual CGRA called Dynamically Reconfigurable Resource Array (DRRA).

II. Preliminaries

In this paper we will attempt to implement phenomena known as Spiking Timing Dependent Plasticity (STDP) using Spiking Neural Networks (SNN). In order to verify our results the CGRA platform we used named Dynamically Reconfigurable Resource Array (DRRA). The computational systems based on SNN is getting critical requires improved and faster computations. The CGRA becomes a promising solution for providing a scalable and robust interconnection fabric. Therefore before explaining the hardware implementation we will briefly describe the SNN and the STDP functionality.

A. Spiking Neural Network (SNN)

Spiking Neural Networks is the latest generation of neural networks models [9]. Spiking neurons emulates the behavior of biological neurons that communicate with electrical pulses or spikes. The spiking neuron is modeled by a differential equation that traces the changes of the potential of a neuron over time [10]. The simplest and the most widely used Leaky Integrate and Fire model is presented in Equation 1. The potential V integrates input spikes I and leaks over time with $-bV$ component. Coefficient a determines equilibrium point and coefficient b the speed of leakage. When the threshold potential is reached a neuron output a spike and its potential is reset. The neurons are connected in one-to-many fashion. Each time a neuron produces a spike, it should be delivered to all the connected neurons. Each connection between neurons has a weight that defines the input to the neuron model from a particular spike. Spikes carry only events, synapses are responsible for generating weighted inputs and for learning. Learning is performed by adjusting synaptic weights. The most widely used learning methods for SNN are Hebb's based rules. Hebb postulated that when one neuron assists in firing another, the synaptic weight between them is strengthened [11]. One of the most widely used models of Hebbian learning is exponential STDP rule shown in Equation 2 [10]. This rule increases a synaptic weight if the time difference between an output (post- synaptic) and an input (pre- synaptic) spikes $\triangle t$ is positive (i.e. when a pre- synaptic spike arrives before a post- synaptic one) and vice versa.

$$dv/dt = I + a - bV \tag{1}$$

$$\begin{cases} A_{+}exp(-\triangle t/\tau_{+}), & \text{if } \triangle t > 0. \\ A_{-}exp(-\triangle t/\tau_{-}), & \text{if } \triangle t < 0. \end{cases} \tag{2}$$

B. Spike-Timing-Dependent Plasticity (STDP)

STDP is a biological process that manipulated the strength of connections between various neurons in the brain. The connection strengths are adjusted depending on the relative timing of a particular neuron's output and input action potentials (or spikes). Further details about STDP can be found in [12].

III. System overview

To evaluate experimentally efficiency of Spiking Neural Networks (SNNs) on a CGRA, we have chosen the Dynamically Reconfigurable Resource Array (DRRA) Shami2012. Nevertheless, it seems that the results are essentially applicable to most grid based CGRAs as well. DRRA is a CGRA template that targets DSP applications. As depicted in Fig. 1, it is composed of three main components: (i) System Controller, (ii) Storage layer (DiMArch), and (iii) Computation layer. Table I briefly describes the basic functionality of these components. The system controller is the general manager of the system that performs functions like application mapping, power management, etc. For the context of this paper, it generates the configware for different neural con-connections and sends it to the computational layer. The storage layer feeds both the configware and data to the computational layer. Because DRRA can host simultaneously multiple applications, for each application a separate partition can be created by the system controller in the DRRA storage and computation layers. Before explaining how neural networks are realized, we will first detail the DRRA architecture.

TABLE I
Local controller functionality

Component	Functionality
DRRA system controller	(i) Send configware to DRRA computation layer (ii) Create memory partitions for each application
DRRA storage layer	Store date for DRRA computation layer
DRRA computation layer	Perform computations

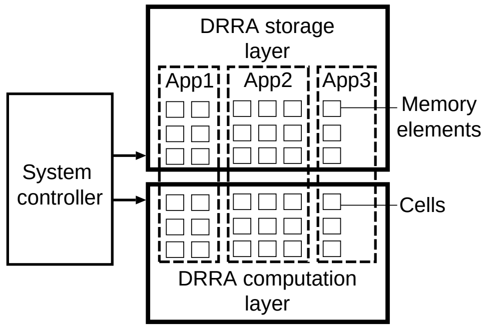

Fig. 1. DRRA system overview

A. System Controller

In this paper, the system controller is used to generate the configware that determines the connectivity between different neurons. Fig. 2 illustrates the system level view of DRRA. The configware is generated by a separate thread (called configware thread) implemented in software on the LEON3 processor. The configuration thread receives the information about the topology and multiplexing technique employed and generates the configware [13].

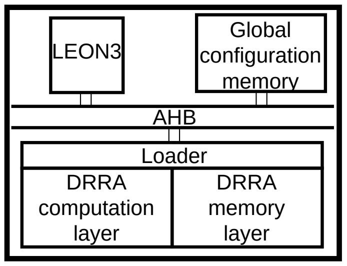

Fig. 2. DRRA system Overview

B. DRRA Computation Layer

DRRA computational layer, shown in Fig. 3, performs computations on data. The computational layer of DRRA contains four elements: (i) register files (reg-files), (ii) morphable Data Path Units (DPUs), (iii) circuit switched interconnects (SBs), and (iv) sequencers. The reg-files are used to store for the DPUs. DPUs are the main functional blocks that perform computations. SBs are the switches that provide circuit switched inter-connectivity between different DRRA components. The sequencers serve as distributed configuration memory for DRRA. Each sequencer holds the configware that corresponds to the functionality of reg-files, DPUs, and SBs. A sequencer stores up to 64 36-bit instructions that determine the functionality of all other elements in its own *cell*. Where a *cell* consists of reg-file, DPU, SBs, and a sequencer, all having the same row and column number as a given cell.

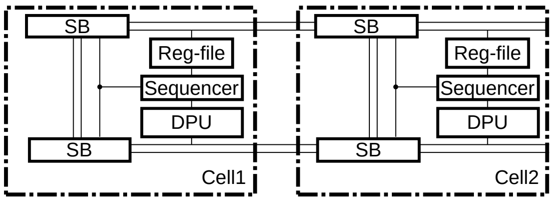

Fig. 3. DRRA computation Layer

C. DRRA Storage layer (DiMArch)

To prevent memory bottlenecks, DRRA boosts a distributed memory called DiMArch [14], [15]. DRRA was designed to host multiple applications with potentially different memory to computational ratio. To efficiently host multiple applications with arbitrary memory to computational ratios, it allows to create a separate memory partition for each application [14]. As shown in Fig. 4, DiMArch is a 2-dimensional array of *memory tiles*. Depending on their function, the tiles are classified into two types: (i) Configuration Tile (ConTile) and (ii) Storage Tile (STile). The memory tiles present in the row, adjacent to the DRRA computation layer, are called ConTiles. The ConTiles manage all data transfers and contain five components: (i) SRAM, to store data for computational layer, (ii) an address generator to provide data from appropriate addresses, (iii) a crossbar, to handle data transfers between tiles, (iv) an Instruction Switch (iSwitch), to handle the transfer of control instructions between tiles [16], and (v) a DiMArch sequencer, to store the sequence in which data will be transferred to the DRRA computational layer. The memory tiles present in rows, nonadjacent to the DRRA computational layer, are called STiles. They are mainly meant for data storage and therefore do not contain the DiMArch sequencer.

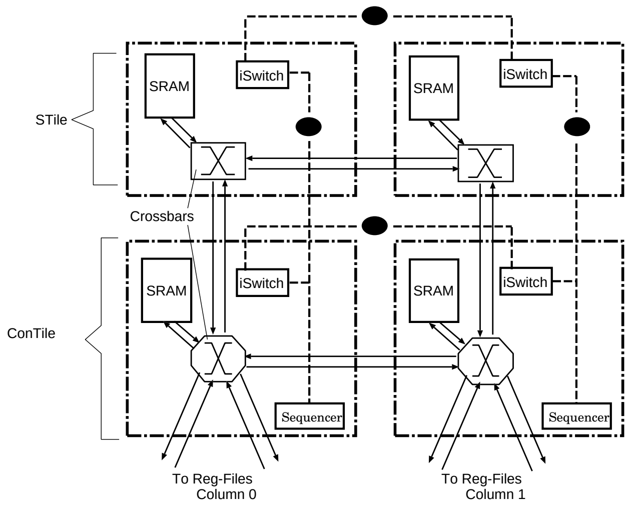

Fig. 4. DRRA storage Layer

D. DRRA programming flow

As shown in Figure 5, DRRA is programmed in two phases (off-line and on-line) [2]. The configware (binary) for commonly used DSP functions (FFT, FIR filter e.t.c.) is written either in VESYLA (HLS tool for DRRA) and stored in an off-line library. For each function, multiple versions, with different degree of parallelism, are stored. The library, thus created, is profiled with frequencies and worst case time of each version. To map an application, its (simulink type) representation is fed to the compiler. The compiler, based on the available functions (present in library) constructs the binary for the complete application (e.g. WLAN). Since the actual execution times are unknown at compile time, the compiler sends all the versions (of each function), meeting deadlines, to the runtime configware memory. To reduce memory requirements for storing multiple versions, the compiler generates a compact representation of these versions. Details of compression algorithm and how it is unraveled are given in [2]. The compact representation is unraveled (parallelized/serialized) dynamically by the runtime resource manager (running in Leon3 processor).

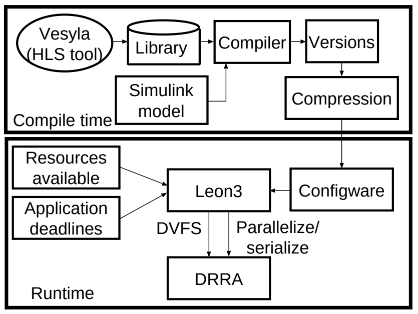

Fig. 5. DRRA programming flow

IV. Modelling spiking neural networks on DRRA

Table II summarizes how various spiking neural network components, described in Section II-A, are implemented on DRRA platform. To mimic the neuron functionality, we have enhanced the functionality of the Data Path Unit (DPU) and the reg-files. Using the reg-file to receive data from other neurons, the DPU performs the computationtions on the received data. Since the spiked neural networks require timing information, we have added a counter and a SNN unit (explained later in Section IV-B). To allow communications between neurons, we exploit the circuit network provided by DRRA, unchanged. Finally, since the spiked neural networks require point to point connectivity (while DRRA register files have only two ports), we have used sequencers to implement point to point communication between multiple neurons (the architecture is discussed in detail in Section IV-A).

TABLE II
Summary of how various neural network components are realized

Neural network entity	DRRA implementation
Neuron/Synapse	DPU + Regfile
Inter neural communications	Circuit switched interconnect+ sequencer

A. Inter Neural Communication

Neural networks require point to point communication between multiple neurons. To implement these connections on DRRA, we had to consider two architectural proper-ties: (i) every DRRA component has only two read/write ports and (ii) a DRRA component can be directly connected to a component at most three hops away. Since these architectural characteristics were designed after a careful evaluation of area/power trade-off, we decided not modify them. The point to point connectivity was realized by using time division multiplexing. In the proposed approach a specific time slot was assigned to each pair of neurons. To allow a scalable solution, we have chosen a hierarchical clustered approach shown in Fig. 6.

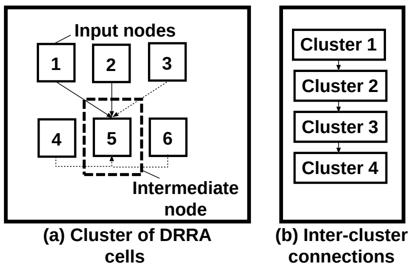

Fig. 6. Inter neural communication realization

Fig. 6 (a) shows a cluster of 6 DRRA cells. In the cluster one of the cells is chosen as an intermediate node. The intermediate node receives data from the 6 cells in the cluster (including itself). To allow connections with 6 cells on a 2-port component, we exploit time division multiplexing combined with partial and dynamic reconfiguration. In the overall process the intermediate cell receives data from 2 cells at a time and then shifts to the other cells. The process continues till the data from all the cells is received. Once every cluster has received inputs, the intermediate nodes communicate with each other serially to ensure point to point connectivity. Fig. 7 shows the instructions in DRRA sequencers (needed to implement the time division multiplexing). The figure shows that 5 cycles are needed to collect information from all the DRRA cells in the cluster. It should be noted that 2 additional cycles are required to reconfigure the circuit switched network. The inter cluster communication takes two cycles for reconfiguration and an additional cycle to transmit data.

B. Neuron/Synapses Realization

1) Processing requirements formalization: When a signal is received (or sent) by a neuron additional processing is required. The processing requirements are dependent on the whether a signal is received (called pre-synapse spike) or transmitted also (post- synapse spike). In this section, we will formulate the processing requirements of each spike separately. To formulate processing requirements of pre-synapse spike, let S_{pre} be the received pre-synapse spike received by neuron N_i from a neuron $N_{i?1}$. Upon occurrence of S_{pre}, four operations are performed sequentially: (i) the spike arrival time, T_{pre}, is

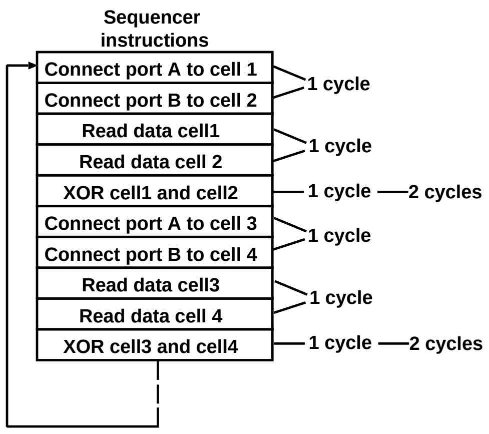

Fig. 7. Sequencer instructions to implement time division multiplexing

stored in $N_{i?1}^{th}$ index of a dedicated pre-synapse array, $A[N]$, where N is the number of neurons, (ii) the time when last post-synaptic spike was transmitted, T_{post}, is retrieved, (iii) the difference between the spike times, $\triangle T = T_{pre}?T_{post}$, is calculated, and (iv) the result calculated by Equation 2 is stored in a weights array $W[N]$. Where τ determines the time intervals when changes of a weight occur and A is constant that determine the maximal change. When the postsynaptic spike, S_{post}, is transmitted three sequential steps follow: (i) the spike transmitting time, T_{post}, is stored$N_{i?1}^{th}$hi?1 index of a dedicated register. (ii) the difference between the spike times, $\triangle T = A[i]?B$, is calculated for each entry in $A[N]$, and (iii) the result calculated by Equation 2 is stored in a weights array $W[N]$.

2) Pre/post-synapse spike realization: Figures 8, 9, and 10 show how the spiking neural network was realized on a DRRA platform. The data path needed to process the pre/post-synaptic spikes is shown in Figures 8 and 9. We have used the reg-files to mimic $A[N]$. Since each reg-files is composed of 64 registers, to realize a system greater than 64 neurons, we have used DiMArch memory (described in Section III). Therefore, for connecting more than 64 neurons an additional overhead of 8 cycles is incurred, to send and receive data from the DiMArch. To implement SNN, requires a divider (see Equation 1) which was missing in the existing DRRA implementation. To realize a divider, we considered two alternatives: (i) to implement the divider using shift registers at the cost accuracy or (ii) to implement a divider in hardware. Since for the realtime robotics (targetted in this paper) speed and power (per computation) are far more significant than silicon efficiency, we opted for a dedicated double precision floating point divider (shown in STDP block Figure 8) to allow quick convergence at low power. In addition, we implemented counters that allowed to transmit the spikes in terms of the time stamps. Finally, the state machine that realizes the sequential steps of pre/post synaptic spikes is implemented.

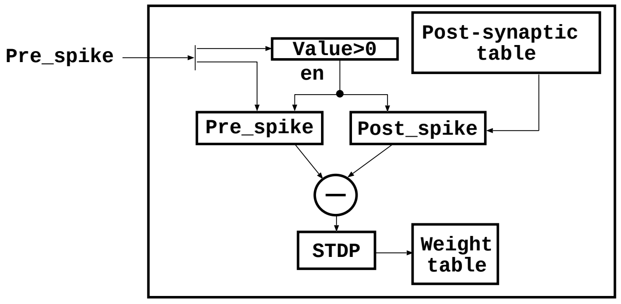

Fig. 8. Dedicated hardware for pre-synaptic spike analysis

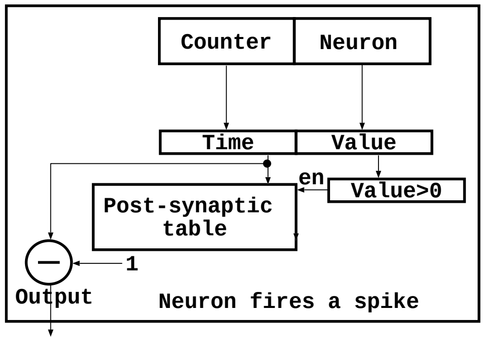

Fig. 9. Dedicated hardware for post-synaptic spike analysis

V. Experimental Results

To investigate the efficacy of the proposed approach, we implemented SNNs on the enhanced DRRA platform. Specifically we evaluated the scalability of the proposed approach and its overheads.

A. Scalability analysis

To determine the scalability of implementing massively parallel neural networks, we measured the cycles needed to realize point to point connectivity using time division multiplexing. We mapped the solution using cluster of different sizes. A cluster is defined as the number of neurons that can communicate directly with each other, in parallel (using space division multiplexing). Since DRRA only permits up to 3 hop connectivity, the maximum cluster size was limited to 5 cells. The scalability was analyzed by calculating the latency for different number of neurons as shown in Fig. 11. As expected, a completely serial implementation (cluster size=1) for 200 neurons required maximum latency of 336000 clock cycles compared to latency with cluster size=5 (that is 67200 for 200 neurons). Considering that DRRA operates at a frequency of 400 MHz, the connectivity for 200 neurons can be achieved in 1.68 msec. Extrapolating the results a connectivity of up to 1000 neurons can be achieved in 4.4 msec. To evaluate additional cells needed for parallel implementation, we analyzed the additional intermediate cells needed for different cluster sizes as shown in Fig. 12. From the Fig. 12, it can be concluded that the minimum area consumption is observed for cluster size1 running 20 parallel neurons and maximum area consumption is observed at cluster size 5 running 200 neurons.

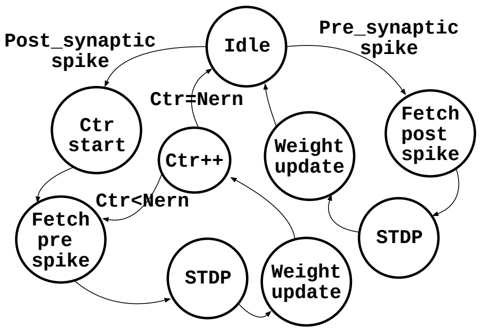

Fig. 10. State machine for pre/post-synaptic spike analysis

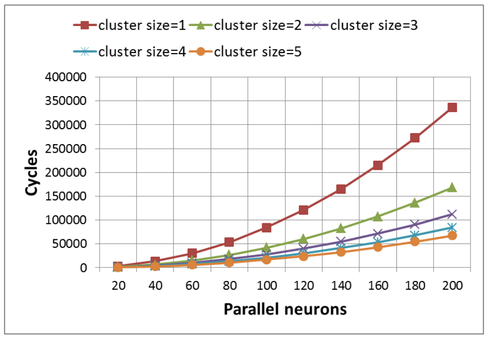

Fig. 11. Parallel neurons computational latency

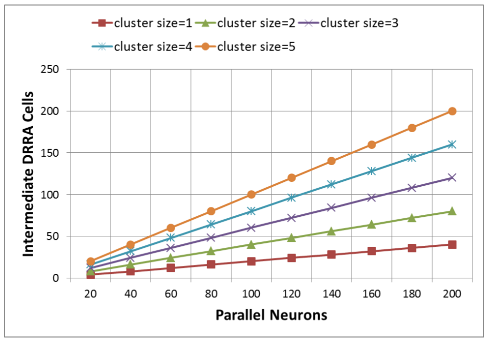

Fig. 12. Cells consumption

VI. Overhead analysis

To estimate the area and power overhead of implementing SNNs, we synthesized the DRRA fabric with enhanced hardware support for 65 nm technology at 400 MHz frequency using Synopsys Design Compiler. Fig. 13 and Fig. 14 illustrate the total area and power usage of the design respectively. The maximum area utilization is for DRRA cells and the area overhead of the system is about 39 % (divider, datapath, and state machine). Similarly, the maximum power consumption is achieved by the DRRA cells and the power consumption for divider, datapath, and state machine is 27 % of the overall design.

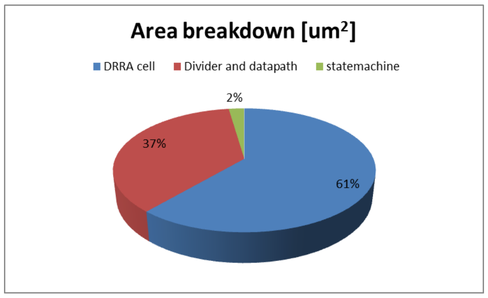

Fig. 13. Area consumption of the design.

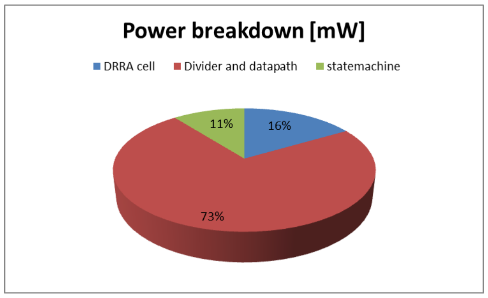

Fig. 14. Power consumption of the design.

VII. Conclusion

In this paper, we have presented the novel idea about the integration of neurons on DRRA platform based on CGRA. We have chosen CGRA as a implementation plat-form because of its potential to dominate its counterpart FPGA in near future. An architecture is presented here is for high performance implementation of Spiked Neural Networks on a CGRA (so far implemented only on FPGAs, ASICs or software) implementation on CGRA leads towards the novelty of this work; we also had exploited the redundancies imposed by the SNNs are quantified on a sample CGRA, called Dynamically Reconfigurable Resource Array (DRRA). The basic idea and theme of this paper is to present the architecture and implementation of neurons on CGRA platform in order to accomplished better performance to achieved higher computational power.

References

[1] P. Garcia, K. Compton, M. Schulte, E. Blem, and W. Fu, “An overview of reconfigurable hardware in embedded systems,” *EURASIP J. Embedded Syst.*, vol. 2006, no. 1, pp. 13–13, Jan. 2006. [Online]. Available: http://dx.doi.org/10.1155/ES/2006/56320

[2] Syed M. A. H. Jafri, K. Paul, A. Hemani, J. Plosila, and H. Tenhunen, “Compact generic intermediate representation (CGIR) to enable late binding in coarse grained reconfigurable architectures,” in *Proc. International Conference on Field-Programmable Technology (FPT),*, Dec. 2011, pp. 1 –6.

[3] D. Alnajjar, H. Konoura, Y. Ko, Y. Mitsuyama, M. Hashimoto, and T. Onoye, “Implementing flexible reliability in a coarse-grained reconfigurable architecture,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,*, vol. PP, no. 99, pp. 1–1, 2012.

[4] M. A. Shami, “Dynamically reconfigurable resource array,” Ph.D. dissertation, Royal Institute of Technology (KTH), Stockholm, Sweden, 2012. [Online]. Available: web.it.kth.se/~hemani/Athesis15.pdf

[5] A. Ghani, T. McGinnity, L. Maguire, and J. Harkin, “Area efficient architecture for large scale implementation of biologically plausible spiking neural networks on reconfigurable hardware,” in *Field Programmable Logic and Applications, 2006. FPL ’06. International Conference on*, 2006, pp. 1–2.

[6] H. Rostro-Gonzalez, G. Garreau, A. Andreou, J. Georgiou, J. Barron-Zambrano, and C. Torres-Huitzil, “An fpga-based approach for parameter estimation in spiking neural networks,” in *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*, 2012, pp. 2897–2900.

[7] J. Harkin, F. Morgan, S. Hall, P. Dudek, T. Dowrick, and L. McDaid, “Reconfigurable platforms and the challenges for large-scale implementations of spiking neural networks,” in *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, 2008, pp. 483–486.

[8] V. Florian, “Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity,” *Neural Comput*, vol. 19, pp. 1468–1502, 2007.

[9] E. Izhikevich, “Simple model of spiking neurons,” *Neural Networks, IEEE Transactions on*, vol. 14, no. 6, pp. 1569–1572, 2003.

[10] S. Song and L. Abbott, “Cortical development and remapping through spike timing-dependent plasticity,” *Neuron*, vol. 32, no. 2, pp. 339 – 350, 2001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0896627301004512

[11] H. D, *The Organization of Behavior: A Neuropsychological Theory, New York: (1949).* Wiley and Sons, 1949.

[12] G. Indiveri, E. Chicca, and R. Douglas, “A vlsi array of low-power spiking neurons and bistable synapses with spike-timing dependent plasticity,” *Neural Networks, IEEE Transactions on*, vol. 17, no. 1, pp. 211–221, 2006.

[13] S. M. A. H. Jafri, O. Ozbak, A. Hemani, N. Farahini, K. Paul, J. Plosila, and H. Tenhunen, “Energy-aware CGRAs using dynamically reconfigurable isolation cells.” in *Proc. International symposium for quality and design (ISQED)*, 2013, pp. 104–111.

[14] M. Tajammul, M. Shami, A. Hemani, and S. Moorthi, “NoC based distributed partitionable memory system for a coarse grain reconfigurable architecture,” in *International Conference on VLSI Design (VLSI Design),*, Jan. 2011, pp. 232 –237.

[15] M. A. Tajammul, M. A. Shami, A. Hemani, S. Moorthi, “A NoC based distributed memory architecture with programmable and partitionable capabilities,” in *Proc. 28th NORCHIP Conf.*, Tampere, Finland, 15–16 Nov. 2010, pp. 1–6.

[16] M. A. Tajammul, M. A. Shami, and A. Hemani, “Segmented bus based path setup scheme for a distributed memory architecture,” in *Proc. IEEE 6th Int. Symp. Embedded Multicore SoCs (MCSoC)*, Sept. 2012, pp. 67–74.