A Parallel Spiking Neural Network Simulator

Kit Cheung [1], Simon R. Schultz [2], Philip H.W. Leong [3]

[1] *Department of Computer Science and Engineering, The Chinese University of Hong Kong*
kcheung6@cse.cuhk.edu.hk

[2] *Department of Bioengineering Imperial College London*
s.schultz@imperial.ac.uk

[3] *Department of Electrical and Information Engineering The University of Sydney*
phwl@ee.usyd.edu.au

***Abstract*—An FPGA-based systolic architecture for the high speed simulation of spiking neural networks is presented. The design is an implementation of Izhikevich's neuron model and employs optimizations for the typical case where neuron activity is low. Since execution time required is related to the activity level, performance of the design can be improved by an order of magnitude.**

I. Introduction

The goal of computational neuroscience is to obtain an understanding of how the brain processes information. Simulations are an important tool for modelling the massively parallel computational processes involved, but the execution time associated with a large scale cortical model is a limiting factor in many cases. In this work, an accelerator for simulating biologically plausible spiking neural networks is presented which allows for the simulation of arbitrary size networks using a single FPGA device.

In a biological neural network, information is exchanged between neurons through the frequency and phase relationships between spikes. At any given time, only a small percentage of neurons are firing (sparse firing) and the actual percentage is known as the activity level. Its value may vary substantially, for instance over the period during which the brain of an organism develops [1], [2]. For high performance, spiking neuron simulations can be optimised to take advantage of this behaviour. The main contribution of this work is an event-driven hardware architecture for simulating a network in which a neuron state is only updated when a neuron spikes. Since spikes occur infrequently, a savings in execution time related to the activity level can be achieved. Thus for a typical case in which the activity level is 1% to 10%, performance is improved almost tenfold.

The architecture is also designed to operate at high frequency and be scalable. This is achieved using a systolic organisation involving mostly local interconnections between processing elements [3]. Multiple FPGAs can be easily connected to produce a larger network and arbitrary size networks can be simulated provided that sufficient memory resources to store the synaptic weights are available.

The rest of this paper is divided into the following sections. In Section II, spiking neural networks and Izhikevich's neuron model are introduced. Section III describes the algorithmic level changes introduced in this paper, namely activity factor optimisation and parallelisation. Section IV explains the architecture and implementation of the simulator. Experimental results are presented in Section V. A comparison with a previous work on spiking neural network implementations is given in Section VI and conclusions concerning this work are given in Section VII.

II. Background

A typical neuron integrates signals from other neurons via synaptic inputs onto its dendrites and soma, applies a non-linearity and generates an output spike if the total exceeds a threshold. While relatively biophysically accurate models such as the Hodgkin-Huxley model have been developed to simulate neurons, it must be noted that these models (which are still phenomenological) are computationally intractable for more than a small number of neurons, and hence not a good choice for large scale spiking networks. Izhikevich proposed a simplified phenomenological model which can reproduce much of the known spiking and bursting behaviour of a wide variety of cortical neurons while allowing simulation of tens of thousands of neurons in real-time on a personal computer [4]. In the description below, the general notation of Thomas and Luk [5] is first introduced and used to describe the neuron dynamics model in an abstract fashion and then the specific implementation of Izhikevich's model is described.

The spiking neural network with N neurons is assumed to be fully connected and hence the output of each neuron i is connected to every other neuron. The synaptic strength of these connections are given by the $N \times N$ matrix $\mathbf{W}$ where $W[i,j]$ is the strength between the output of neuron j and the input of neuron i. Thus $\mathbf{W}[\mathbf{i},:]$ represents the synapses at the input of neuron i, whereas $\mathbf{W}[:,\mathbf{j}]$ represents the synapse values connected to the outputs of neuron j.

Each neuron has its own static parameters and varying state values. The set $\mathbb{P}$ represents the set of possible constant

parameters and $\mathbb{S}$ is the set of neuron states. The set of possible inputs to the neurons is denoted by $\mathbb{R}$.

The neuron updated function $f : (\mathbb{P}, \mathbb{S}, \mathbb{R}) \rightarrow (\mathbb{S}, [0, 1])$. takes as inputs the neuron parameters, states and inputs and produces the next neuron state and binary output.

The computation of a timestep of the simulation is described in Algorithm 1. For each neuron in the network, its input is computed by calculating the dot product of all neuron outputs with the associated synapse value to neuron i, i.e. $i_i = \mathbf{W}[\mathbf{i}, :]\tilde{\mathbf{f}}$. We call this the *ACC phase*. This neuron input is then used along with the neuron's state and parameters to compute the new state and output $(\vec{s_i}, \vec{t_i}) \leftarrow f(\vec{c_i}, \vec{s_i}, i_i)$ in the *CAL phase*.

begin
  // ACC phase ;
  for *i=1..n* **do**
    $i_i \leftarrow W[i, :]\vec{f}$;
  end
  // CAL phase ;
  for *i=1..n* **do**
    $(\vec{s_i}, \vec{f_i}) \leftarrow f(\vec{c_i}, \vec{s_i}, i_i)$;
  end
end

Algorithm 1: Timestep (dense)

Izhikevich's model uses 2 variables to represent the state of a single neuron i, namely its membrane recovery variable $u[i]$ and membrane potential $v[i]$, i.e. $(u[i], v[i]) \in \mathbb{S}$. An additional 5 parameters are used for the configuration of the neurons: a - time scale of u; b - sensitivity of u; c - value of v after the neuron fired; d - value of u after the neuron fired; s - scaling factor for noise input. Hence the neuron parameters are $(a, b, c, d, s) \in \mathbb{P}$. These parameters can be tuned to represent different neuron classes (for instance by fitting membrane potential traces from whole-cell patch clamp recording experiments [6]); in practice, in a large scale simulation each parameter might be selected from a narrow distribution for the respective cell class, thus incorporating network inhomogeneity.

The dynamics of $u[i]$ and $v[i]$ are given by the two dimensional system of ordinary differential equations:

$$du[i]/dt = a(bv[i] - u[i]) \quad (1)$$

$$dv[i]/dt = 0.04v[i]^2 + 5v[i] + 140 - u[i] + J[i] \quad (2)$$

where $J[i] = I[i] + sN(0, 1)$, and $N(0, 1)$ is a zero mean, unit variance normally distributed random input. This noise term simulates the contribution of thalamic noise from non-modelled sources [4].

If the value of $v[i]$ is above 30 mV, the output is set to 1 (otherwise it is 0) and the state variables are reset:

$$\text{if } v[i] \geq 30\ mV \text{ then } \begin{cases} v[i] = c \\ u[i] = u[i] + d \end{cases} \quad (3)$$

III. Optimizations

The time complexity of a spiking neural network simulation lies in the $O(N^2)$ floating point operations required to perform the dot product $i_i = \mathbf{W}[:, \mathbf{i}]\tilde{\mathbf{f}}$ for all neurons. However, the number of nonzero entries in $\vec{f}$ is small. A more efficient sparse implementation uses $setbits()$ which returns the indices of nonzero bits in a binary vector, followed by an empty sentinel (denoted by e).

begin
  // ACC phase;
  for *j in setbits($\vec{f}$)* **do**
    $\vec{i} \leftarrow \vec{i} + W[:, j]$;
  end
  // CAL phase;
  for *i=1..n* **do**
    $(\vec{s_i}, \vec{f_i}) \leftarrow f(\vec{c_i}, \vec{s_i}, i_i)$;
  end
end

Algorithm 2: Timestep (sparse)

Although this has the same $O(N^2)$ complexity as the previous algorithm, the number of floating point operations required is now equal to the number of set bits in $\vec{f}$, and hence reduced by the activity factor. In the implementation described $setbits()$ is implemented using a leading ones detector (LOD).

A parallel implementation of Algorithm 2 is now described. The number of processing elements (PEs) is K and they are connected together in a unidirectional ring. C neurons are handled per PE. A total of KC weight tables each holding $W[i, :]$ is required for accumulation of weights during the ACC phase. The computation of the sparse dot product proceeds in parallel as follows:

1) PE k applies a LOD over the subvector $\vec{f^k} = [f_{Ck}, f_{Ck+1}, \ldots f_{C(k+1)-1}]$ and determines the offset d of the first nonzero bit (indicating neuron $j = Ck+d$ has fired). If the LOD returns e, no updates are applied in the next step below.
2) Each PE computes $\vec{i_k} \leftarrow \vec{i_k} + W[:, j]$ for the C neurons it handles, where i_k are the associated inputs.
3) j is passed to the neighbouring PE and a new j is received. Step 2 is repeated until all K fired neurons have been processed.
4) Repeat from step 1 until all LODs return e.

The parallel implementation method just described has several advantages:

- As mentioned before, the number of cycles required is related to the activity factor.
- The PEs operate in parallel to reduce the execution time by a factor up to K.
- Only a small number of local connections between PEs are required.

IV. Implementation

A. Architecture

The overall datapath of the neural network simulator uses the parallel implementation of Algorithm 2 described in the previous section and is illustrated in Figure 1. Each PE is responsible for computing the response of its C neurons as well as storing their state and synaptic weights. A finite state machine (not shown) is used to control the datapath.

A timestep computation is divided into two phases. The first phase is the ACC phase which performs accumulation of the spikes from other neurons according to Algorithm 2. The CAL phase updates the neurons' states according to Equations 1-3.

A key block in the design is the LOD which determines the first set bit in a binary vector of length C. For example, if $C = 6$ and the vector is "011001" the LOD should return the sequence $0, 3, 4, e$. In our current implementation, a parallel, single cycle implementation of Algorithm 3 is used. Note that successive calls to the LOD return new values of set bits since x is modified on the fly.

Data: x (global variable)
while $(x \neq 0)$ **do**
  // clear all but first unset bit
  $z = x\&(-x)$;
  for $i = 0 \ldots (N-1)$ **do**
    if $(((1 << i)\&z) \neq 0)$ **then**
      $x = x\&(\sim z)$;
      return i;
    end
  end
end

Algorithm 3: Leading ones detector

An example of the accumulation procedure is presented in Figure 2. The following points should be noted:

1) At the beginning of the first pass, the ACC units receive the relative position d of the first non-zero bit of each $\vec{f^k}$ from the LODs $(0, 1, e, 0$ in the example).
2) An offset of Ck for PE k $(+6, +4, +2, +0)$ is added to obtain the absolute fired neuron number $(6, 5, e, 0)$. The corresponding entries in the weight tables for each PE will then be accumulated in the input I_i .
3) The neuron number d is passed to the adjacent PE.
4) Steps 2 & 3 are repeated until all fired neurons in the pass have been processed by each PE. Hence each pass requires a runtime of K cycles.
5) If all LODs return e the ACC phase ends, otherwise go back to Step 1.

The implementation requires a runtime of KA cycles where A is the maximum number of non-zero bits among all of the subvectors $\vec{f^k}$.

B. Fixed Point Precision

Calculations in the proposed implementation are made with 18-bit two's complement numbers which have a sign bit, 9 integer bits and 8 fractional bits. Weights are 9-bit two's complement fractions, having 1 sign bit and 8 fractional bits.

These values were chosen because they fully utilise the 18-bit two's complement multipliers available in the DSP blocks of the Xilinx Virtex-5 FPGA used [7]. Any larger precision would double the DSP utilisation. The range of neuron state variable v is typically between -110 and 390 and is calculated in millivolts.

Weights were chosen to be half of that precision (9-bits) as they are the limiting factor in our design for large networks. We believe this is a reasonable estimate of the precision with which a synaptic weight is defined in the central nervous system. For instance, at the mossy-fibre to parallel fibre synapse in the cerebellum, the most numerous synapse in the brain, there are 200-400 quantal release sites [8] suggesting a resolution of at most 9 bits. In practice, a more limited range is likely to be used. It should also be noted that this is a generalisation that can only be carried so far - for instance, the calyx of Held, a synapse in the auditory brainstem, might potentially have greater resolution [9].

C. Memory Organization

Three different types of data values need to be stored in the CAL unit for an implementation of Izhikevich's model. These are the neuron states (v and u), the neuron parameters (ab, $1-a$, c, d) and synaptic strength $\mathbf{W}$. As suggested by Thomas and Luk [5], ab and $1-a$ are precomputed so to facilitate the evaluation of Equation 1 so $u \leftarrow (ab)v + (1-a)u$ can be computed instead of $u \leftarrow a(bv - u) + u$.

The synapse values dominate the memory requirements since it is $O(N^2)$ whereas the others are $O(N)$. For this reason, the $\mathbf{W}$ values are stored in dedicated Block RAMs [7] whereas the states and parameters are stored in Distributed RAM [7].

Each neuron has its own table of depth N each storing the synaptic strength $\mathbf{W}[\mathbf{i}, :]$, where N is the number of neurons to be simulated. Since the size of Block RAMs is 18Kbit [7], each can hold at most 2048 9-bit synaptic values, hence at least $\lceil \frac{N^2}{2048} \rceil$ Block RAMs are required in the full implementation.

D. ACC Unit

During the CAL phase, spike accumulation is achieved by accumulating one entry of each synaptic strength table every cycle. Fig. 3 shows the architecture of the ACC unit. The memory tables $\mathbf{W}[\mathbf{i}, :]$ for $i \in [kC + 1..k(C + 1)]$ are stored in the ACC unit of PE k.

The Block RAM receives an address from LOD in the first cycle, and then receives address from the other PEs for the remaining K-1 cycles. A counter is used to record the status of the accumulation and determine the source of address of the Block RAM. The offset mentioned in step 2 of section IV-A is calculated using the counter and is added to the address. The counter also determines when to obtain the next address from LOD.

The AND-ed output of the e's from the LODs disables the accumulators. When asserted it indicates the end of the

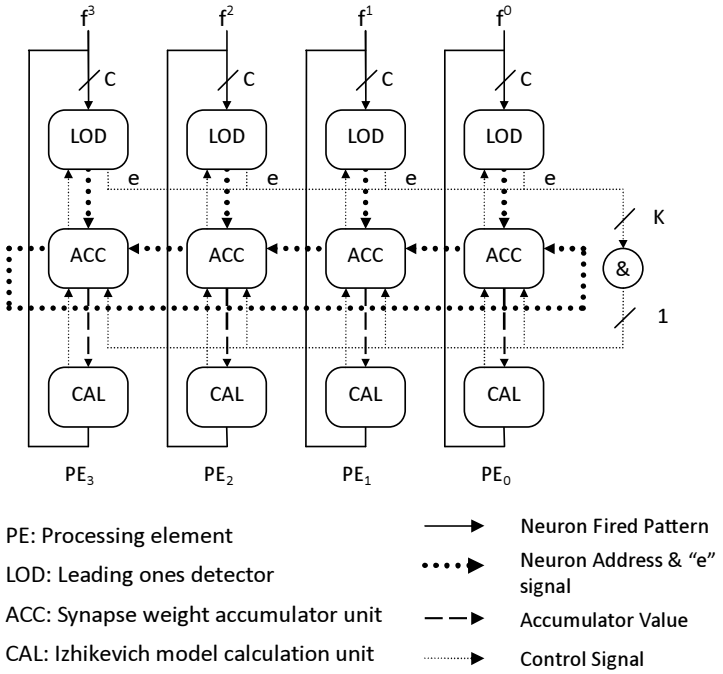

Fig. 1. Datapath of the spiking neural network simulator. Four PEs $K = 4$ are shown, each handling C neurons. Thus a total of $4C$ neurons can be simulated.

ACC phase and hence the accumulator will stop accumulating spikes. This also causes the CAL phase to commence. At the end of the CAL phase, the accumulators will be reset and the next ACC phase will start.

E. CAL Unit

The CAL unit is pipelined and produces an output every cycle with a latency of 6 cycles. The datapath is shown in Fig. 4. A 0 to $C-1$ counter is used to select accumulator values in the ACC unit and the read address of the neuron parameter memory. The neuron update pipeline takes the accumulator value, neuron parameters and states as input and produces a new u and v value after 6 cycles which is then written into the neuron state memory. The write address of this memory is connected to the counter value minus 6 since the output is delayed by 6 cycles.

The implementation of Equations 1-3 requires 5 multiplications, one of them being with a constant which is implemented without using a DSP block. Hence the unit requires 4 multipliers in total. The number of DSP blocks available is the limiting factor for K, the number of PEs in the device.

F. Gaussian Random Number Generator

A Gaussian random number generator (GRNG) models the thalamic noise input explained in Section II. Many different schemes are possible [10] but in this particular application, a GRNG is required per PE so small resource utilisation is more important than accuracy in the tails of the distribution. Since the number of memory and DSP blocks can limit the size of the network which can be simulated, the GRNG should not use these resources.

In our implementation, the Central Limit Theorem approach which involves summing multiple uniform random number generators to approach a Gaussian distribution was used. The GRNG is formed by summing four linear feedback shift registers (LFSR) based uniform random number generators, which in turn were chosen because they occupy the smallest amount of hardware resources. Its seed is related to the ID of the PE, making the simulations repeatable. Of course, it can be seeded with a random number should that be desirable.

The following equation is used to generate an N(0,1) random number [10]:

$$R = \sqrt{\frac{12}{L}}(\sum_{i=1}^{L} U_i - \frac{L}{2}) \tag{4}$$

where U_i is the output of i-th LFSR-based random number generator and $L = 4$ in our implementation.

G. Multiple-FPGA Network

In a multiple-FPGA network, a single unidirectional connection between adjacent FPGAs of width $\lceil log_2 C \rceil$ is required for the neuron address where C is the number of neurons handled by each processing unit. For large networks, the synaptic weight storage is the limiting factor, and external memory can be considered. The memory bandwidth is highest for the ACC phase and hence may limit the speed of simulation. For each FPGA, a total of $\frac{9NR}{N_F}$ bits need to be transferred per cycle, where R is the clock rate of the design and N_F is the total number of FPGAs used. Figure 5 shows how a larger network can be built by concatenating 4 FPGAs as a ring array where each of the FPGAs stores 2 PEs.

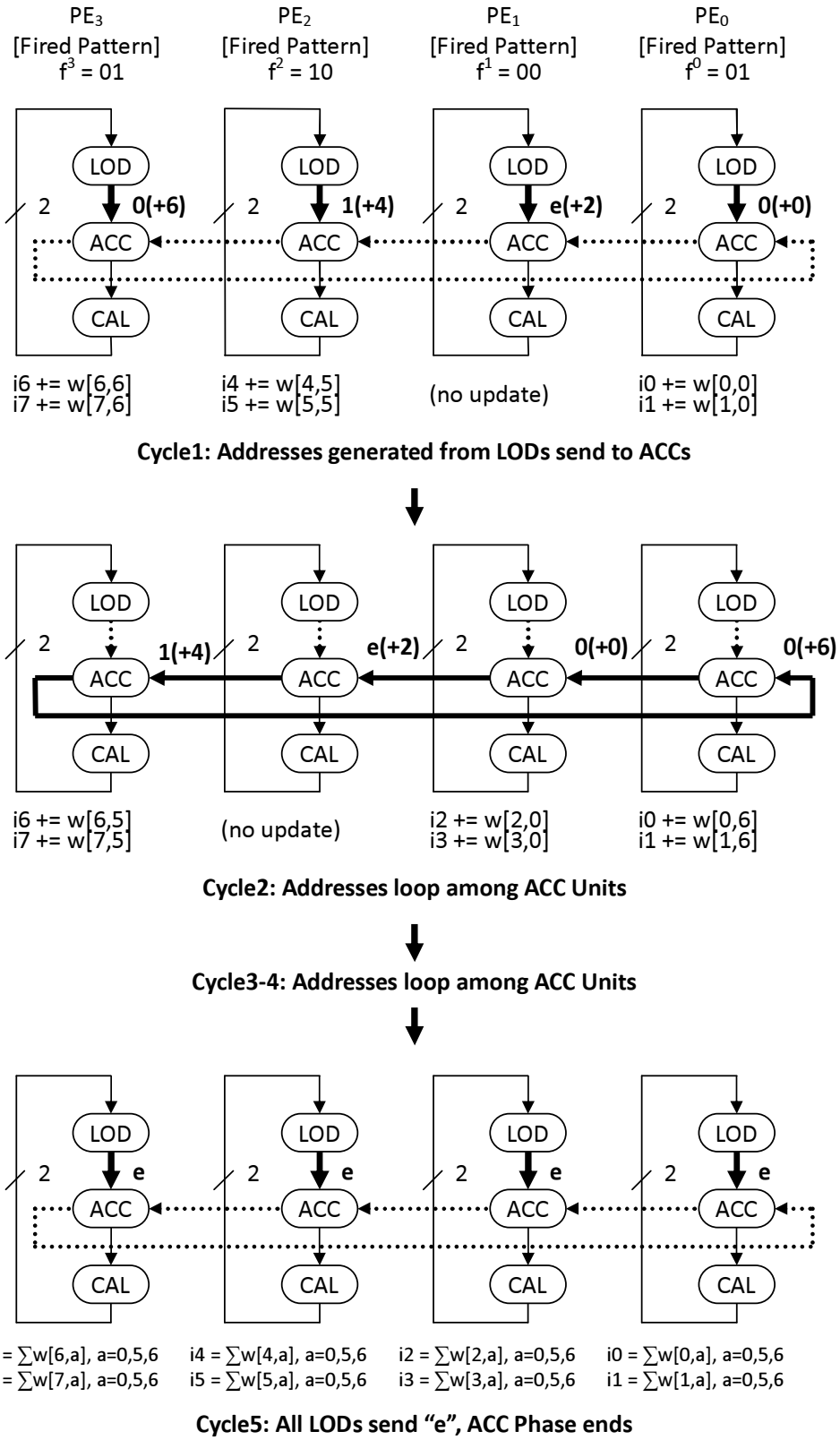

Fig. 2. ACC computation for an 8-neuron simulation. Neurons 0, 5 and 6 have fired and I_i values are accumulated in the 4 PEs, each one handling two neurons.

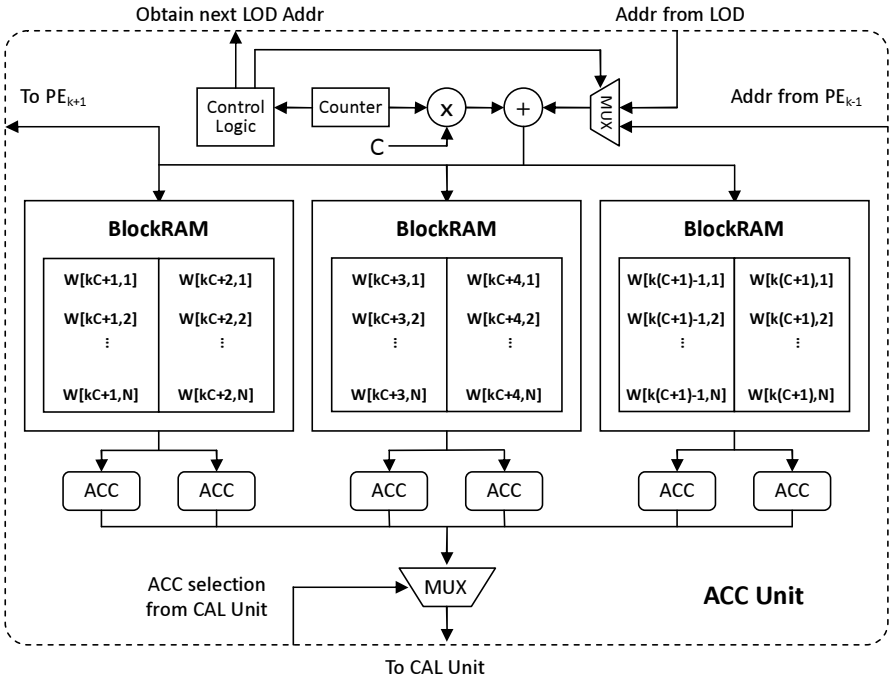

Fig. 3. Simplified architecture of ACC Unit, each Block RAM storing 2 weight tables (register and control logic omitted)

V. Results

A fully-connected 800-neuron network was simulated using the Xilinx Virtex-5 XC5VLX155T target device. This medium-size device has 424 18Kbit single port Block RAMs (or 212 36Kbit dual port Block RAMs) and 128 DSP blocks. Hence this device supports at most $K = \frac{128}{4} = 32$ PEs and $N = 848$ neurons.

Xilinx ISE 9.2i was used to generate the implementation. VHDL generic parameters are used throughout the design so that most parameters including K and C can be easily changed. An implementation using $K = 32$ and $C = 25$ was made and a summary of the resource utilisation is given in Table I. A general estimate of the resource utilisation ratio of major components is given in the table along with a breakdown of the percentage used by each of the LOD, GRNG, ACC and CAL units.

The critical path is that of the LOD described in Section III. Although this is reasonably well balanced with other near-critical paths in the current implementation, for large C the LOD may limit the performance of the system. As the design does not require a new value from the LOD every cycle, a multi-cycle implementation could be used to improve the clock frequency of the LOD.

In the design presented execution time is a function of the number of PEs used K. Since the ACC phase requires $K\lceil\frac{AN}{K}\rceil$ cycles per timestep and CAL requires $\frac{N}{K} + 12$, the total execution time per timestep is approximately equal to their sum. Figure 6 shows the effect of increasing K. Initially, the number of cycles required reduces dramatically due to the increased parallelism. However, this value saturates at approximately $K = 10$, after which hardware resources are increased with little improvement. The optimal value of K is dependent on the dynamics of the network being simulated.

Resource	Used (%)
Logic Blocks (CLB)	54540 (56%)
Flip Flops (FF)	31960 (32%)
LUT	35229 (36%)
DSP	128 (100%)
Block RAM	208 (98%)
Clock Rate	110.47MHz

TABLE I
Resource utilisation for the $K = 32, C = 25$ case.

Resource	Complexity	LOD	GRNG	ACC	CAL
Flip Flop	$\propto K, C$	2%	16%	43%	39%
LUT	$\propto K, C$	14%	5%	55%	26%
Block RAM	$\propto N^2$	0%	0%	96%	4%
DSP	$\propto K$	0%	0%	0%	100%

TABLE II
Resource utilisation for the general case. Some neuron states and parameter are stored in both block RAM and distributed RAM to save LUT usage.

A randomly connected network of 800 neurons was simulated using the same parameters as Reference [4]. A 4:1 ratio of excitatory to inhibitory neurons was used. Excitatory neurons were created by initialising the ith neuron as $(a_i, b_i) = (0.02, 0.2)$ and $(c_i, d_i) = (-65, 8) + (15, -6)U_i^2$ where U_i is a uniformly distributed random variable in [0,1]. Inhibitory neurons initialised as $(a_i, b_i) = (0.02, 0.25)$ and $(c_i, d_i) = (0.08, -0.05)U_i + (-65, 2)$. The randomness introduced by U_i allows the neuron population to have different dynamics and emulate different cell types. They are regular

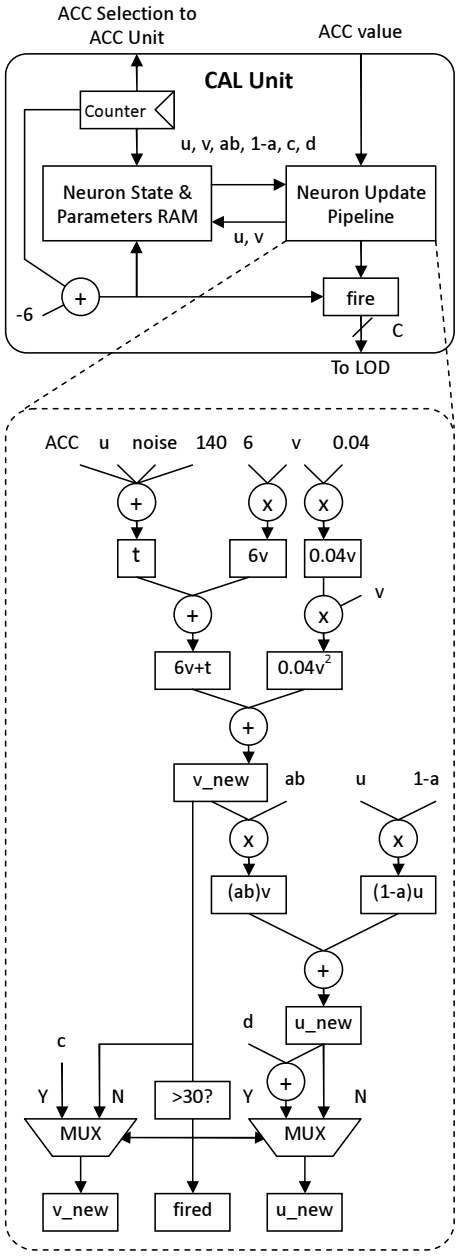

Fig. 4. Datapath of the CAL Unit (pipeline stages not shown).

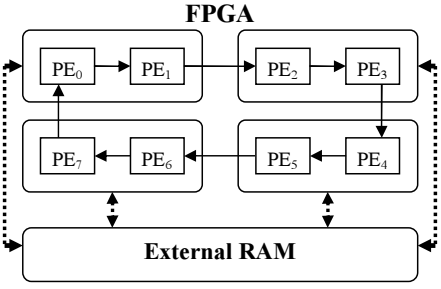

Fig. 5. A 8-PE Network using 4 FPGAs

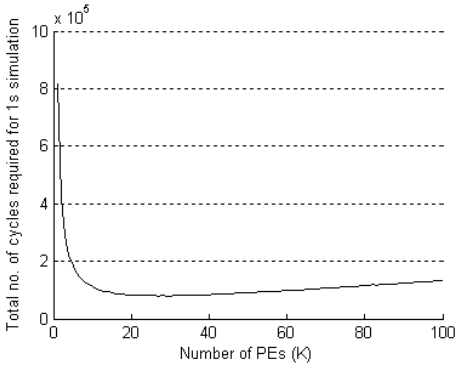

Fig. 6. Graph showing cycles per timestep as a function of K, the number of PEs used.

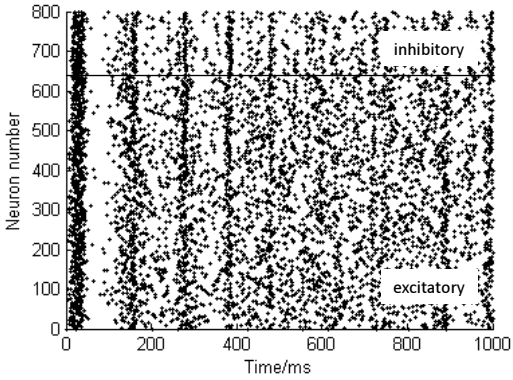

Fig. 7. Simulation showing synchronisation of firing activity in a spiking neural network simulation involving 800 neurons.

cells when $U_i = 0$ and chattering cells for $U_i = 1$ with a bias towards the regular cells. Synaptic connections to excitatory neurons were positive uniform random numbers in the range [0,0.5] and inhibitory neurons had negative strengths in the range [-1,0].

A $1\ s$ simulation of the system with a timestep of $1\ ms$ was made and the resulting output is shown in Figure 7. The characteristic $10\ Hz$ alpha and $40\ Hz$ gamma rhythms which are similar to that seen in the mammalian cortex are present. This is due to neurons self-organising into assemblies with collective rhythmic behaviour [4].

A total of 6107 firings in 1000 timesteps were observed, supporting our assertion that the activity level is low. One or more neuron firings were observed in 97% of timesteps, and this limits the savings which can be enjoyed. Figure 8 shows how the number of cycles per timestep changes with time in the simulation. Timestep with heavy firing activity indicated in Figure 7 takes more cycles to simulate. Total execution time is approximately $0.73\ ms$ which amounts to a speedup of $1370\times$ over real time.

VI. Related Work

Research on spiking neural networks have traditionally relied on simulations on conventional and parallel computers and a survey of FPGA-based implementations is available in Reference [11].

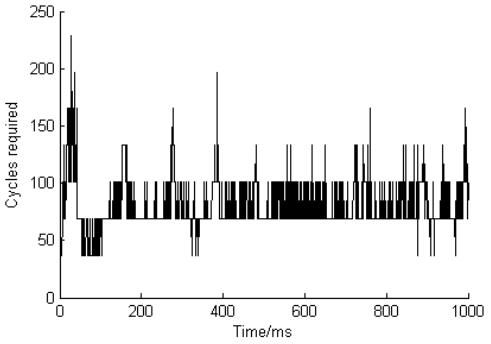

Fig. 8. Graph showing the number of cycles required per timestep for the SNN simulation of Figure 7.

In 2006, Ros et. al. described a platform for simulating arbitrary networks of spiking neurons which used an event-driven communications scheme to connect a software-based model of spike routing and learning with FPGA-based computation of the neural model [12]. The approach was developed to model small to medium sized networks rather than large ones, and the models were of much higher complexity than used in this work, with correspondingly lower performance.

Our design is most similar to a recent pipelined implementation of Izhikevich's model by Thomas and Luk [5] which does not optimise for activity level. The following differences between the two implementations are noted:

- Their implementation achieved a speedup of $148\times$ real time for $N = 800$, an order of magnitude slower than the one described in this paper. This is because they require N cycles per timestep whereas in our design, $K\lceil\frac{AN}{K}\rceil + N/K + 12$ are required.
- Approximately 55000 CLBs are used in this design for $N = 800$, while their implementation only requires 33000 CLBs for $N = 1000$. In terms of the largest network which can be simulated, both designs are limited by the amount of Block RAM resources required to store the synaptic values.
- Thomas and Luk's implementation uses floating point arithmetic whereas our design uses fixed point which we believe is more biologically plausible as discussed in Section IV-B.
- Finally, it may be difficult to scale their design to a multi-FPGA system whereas our design can be easily extended by virtue of its ring connected architecture.

VII. Conclusion

The performance of spiking neural network simulations can be improved dramatically by utilising the fact that neurons spike infrequently. An event-based architecture for implementing spiking neural network simulations was proposed and can achieve approximately $1400\times$ real-time performance which is an order of magnitude faster than the best previously reported design.

In future work we intend to extend its functionality so it can be used for large-scale simulations. For example, we hope to use a model proposed by Izhikevich in a newer paper where the entire cortex is simulated [13].

Ultimately, of course, we would like to use reconfigurable computing to simulate networks which are intractable with other technologies. We will also explore architectures which allow for optimum use of hardware to be maintained in the face of systematic changes during development. This is likely to be an important issue, because as larger scale cortical models are built, they will not come "pre-wired" as in the case presented. The problem of wiring them up to perform particular tasks is going to be more and more a critical issue. The way nature solves this problem involves initially allocating a large number of synapses, which are eliminated during sensory experience during the "critical" developmental period. At the same time, the firing rates of individual neurons begin quite low, but increase during development. Some other properties of neurons (such as temporal integration) also change. FPGA architectures are thus ideally suited to this job, as they (through reprogramming, perhaps periodically) allow the architecture to be adjusted to maintain an efficient architecture for the balance of number of synapses and activity level found at each stage in the developmental process.

Acknowledgment

The authors gratefully acknowledge support from the Research Grants Council of the Hong Kong Special Administrative Region, China (Earmarked grant CUHK414308).

References

[1] C. Blakemore, "Maturation of mechanisms for efficient spatial vision," in *Vision: coding and efficiency*, C. Blakemore, Ed. Cambridge University Press, 1990, pp. 254–266.

[2] N. C. Rust, S. R. Schultz, and J. A. Movshon, "A reciprocal relationship between reliability and responsiveness in developing visual cortical neurons," *Journal of Neuroscience*, vol. 22, pp. 10 519–10 523, 2002.

[3] H. T. Kung and C. E. Leiserson, "Systolic arrays (for VLSI)," in *Proc. SIAM Sparse Matrix Symposium*, 1978, pp. 256–282.

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

[5] D. Thomas and W. Luk, "FPGA accelerated simulation of biologically plausible spiking neural networks," in *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2009.

[6] E. de Lange and M. Hasler, "Predicting single spikes and spike patterns with the Hindmarsh-Rose model," *Biological Cybernetics*, vol. 99, pp. 349–360, 2008.

[7] Xilinx Inc., *Virtex-5 FPGA Data Sheet*, 2009. [Online]. Available: www.xilinx.com

[8] P. B. Sargent, C. Saviane, T. A. Nielsen, D. A. DiGregorio, and R. A. Silver, "Rapid vesicular release, quantal variability, and spillover contribute to the precision and reliability of transmission at a glomerular synapse," *Journal of Neuroscience*, vol. 25, pp. 8173–8187, 2005.

[9] T. Sakaba, R. Schneggenburger, and E. Neher, "Estimation of quantal parameters at the calyx of held synapse," *Neuroscience Research*, vol. 44, pp. 343–356, 2002.

[10] D. B. Thomas, W. Luk, P. H. Leong, and J. D. Villasenor, "Gaussian random number generators," *ACM Comput. Surv.*, vol. 39, no. 4, p. 11, 2007.

[11] L. P. Maguire, T. M. McGinnity, B. Glackin, A. Ghani, A. Belatreche, and J. Harkin, "Challenges for large-scale implementations of spiking neural networks on FPGAs," *Neurocomput.*, vol. 71, no. 1-3, pp. 13–29, 2007.

[12] E. Ros, E. M. Ortigosa, R. Agís, R. R. Carrillo, and M. Arnold, "Real-time computing platform for spiking neurons (RT-spike)," *IEEE Transactions on Neural Networks*, vol. 17, no. 4, pp. 1050–1063, 2006.

[13] E. M. Izhikevich and G. M. Edelman, "Large-scale model of mammalian thalamocortical systems," *Proc. Natl. Acad. Sci. USA*, vol. 105, pp. 3593–8, 2008.