

# A Parallel Spiking Neural Network Simulator

Authors removed for blind review

**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 proportional to the activity level, performance of the design can be improved by an order of magnitude. A parameterised implementation of this architecture is also presented which is scalable and flexible, supporting different precisions, wordlength and network sizes. Results are given which demonstrate the utility of this approach.

## 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), the actual percentage being known as the activity level. This activity level may vary substantially, for instance over the period during which the brain of an organism develops [1]. It is thus important to develop spiking neuron simulation architectures which can be optimised for a given activity level. 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 proportional to the activity level can be achieved. Thus for a typical case in which the activity level is 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 [2]. 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. Finally, conclusions concerning this work are given in Section VI.

## 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 [3]. In the description below, the general notation of Thomas and Luk [4] 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  $\mathbf{W}[i, :]$  represents the synapses at the input of neuron  $i$ , whereas  $\mathbf{W}[:, j]$  represents the synapse values connected to the outputs of neuron  $j$ .

Each neuron has its own static parameters and varying state values. The set  $\mathcal{P}$  represents the set of possible constant parameters and  $\mathcal{S}$  is the set of neuron states. The set of possible inputs to the neurons is denoted by  $\mathcal{R}$ .

The neuron updated function  $f : (\mathcal{P}, \mathcal{S}, \mathcal{R}) \rightarrow (\mathcal{S}, [0, 1])$ . 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.  $i_i = \mathbf{W}[:, i]\mathbf{f}$ . 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  $(\bar{s}_i, \bar{t}_i) \leftarrow f(\bar{c}_i, \bar{s}_i, i_i)$  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)

```

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)

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.  $(u[i], v[i]) \in \mathcal{S}$ . 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 a normally distributed random input which is added to each neuron and simulates the contribution of noise from non-modelled sources [3]. Hence the neuron parameters are  $(a, b, c, d, s) \in \mathcal{P}$ . These parameters can be tuned to represent different neuron classes (for instance by fitting membrane potential traces from whole-cell patch clamp recording experiments [5]); 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:

$$\begin{aligned} du[i]/dt &= a(bv[i] - u[i]) & (1) \\ dv[i]/dt &= 0.04v[i]^2 + 5v[i] + 140 - u[i] + I[i] & (2) \end{aligned}$$

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

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

### 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  $i_i = \mathbf{W}[:, i] \vec{f}$  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 a binary vector which have set bits, followed by an empty sentinel (denoted by  $e$ ).

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 in this paper,  $setbits()$  is implemented using a leading ones detector (LOD).

A parallel implementation of the 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  $\vec{f}^k = [f_{Ck}, f_{Ck+1} \dots f_{C(k+1)-1}]$  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 each of the  $C$  neuron that 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 proportional to the activity factor; the PEs operate in parallel to reduce the execution time by a factor of  $K$ ; and only a small number of local connections between PEs are required.

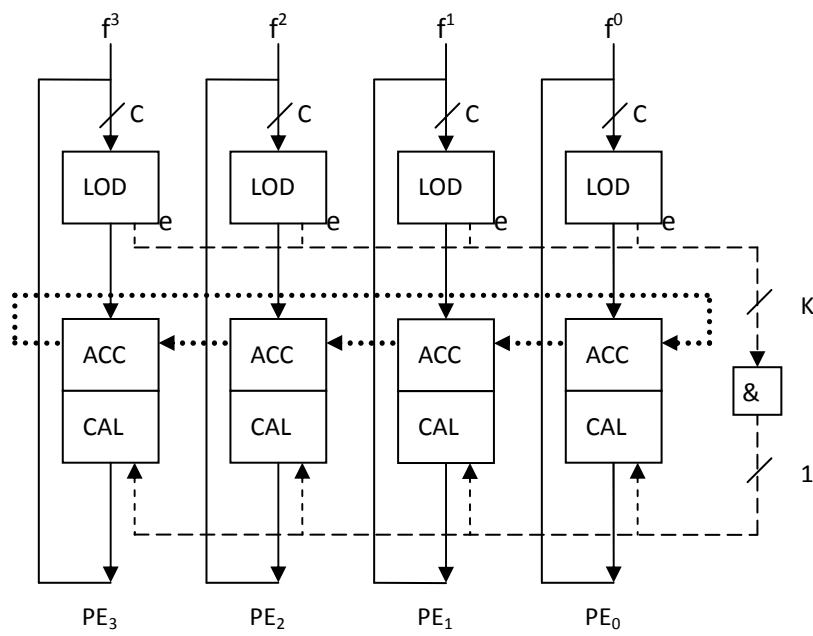
## IV. IMPLEMENTATION

### 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. This block 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 0, 3, 4,  $e$ . The current implementation uses a parallel, single cycle implementation of the algorithm below. Note that successive calls to the LOD return new values of set bits due to  $x$  being a global variable.



PE: Processing element                       $\longrightarrow$  Data Flow  
LOD: Leading ones detector                 $\cdots\cdots\cdots\longrightarrow$  Neuron Address Flow  
ACC: Synapse weight accumulator unit    $-\cdots-\cdots\longrightarrow$  State Control Flow  
CAL: Izhikevich model calculation unit

Fig. 1. Datapath of spiking neural network simulator. In the figure, four PEs  $K = 4$  are shown, each handling two neurons  $C = 2$ . Thus a total of  $N = KC = 8$  neurons can be simulated.

Although this implementation is sufficient for the values of  $C$  used in this work, for large  $C$  the LOD may affect the performance of the system. As the simulation does not require a new value from the LOD every cycle, a multi-cycle implementation could also be considered.

```

Data: x (global variable)
while (x ≠ 0) do
  // clear all but first unset bit
  z = x & (-x);
  for i = 0 ... (N - 1) do
    if (((1 << i) & z) ≠ 0) then
      x = x & (~ z);
      return i;
    end
  end
end
end

```

Algorithm 3: Leading ones detector

An example of the accumulation is presented in Figure 2. Some points about its implementation are:

- 1) At the beginning of the first pass, the ACC units receive the relative position of the first non-zero bit of each  $f^k$  from the LODs (0, 1, e, 0 in the example).
- 2) An offset (+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  $I_i$ . Since the offset can be predicted by the ID of the PE, it can be implemented as a counter local to the PE.

- 3) The relative address 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 to most efficiently use the 18-bit two's complement multipliers available in the DSP blocks of the Xilinx Virtex-5 FPGA used [6]. Any larger precision would double the DSP utilisation. The range of neuron state variable  $v$  is typically between -110 and 390 and is calculated

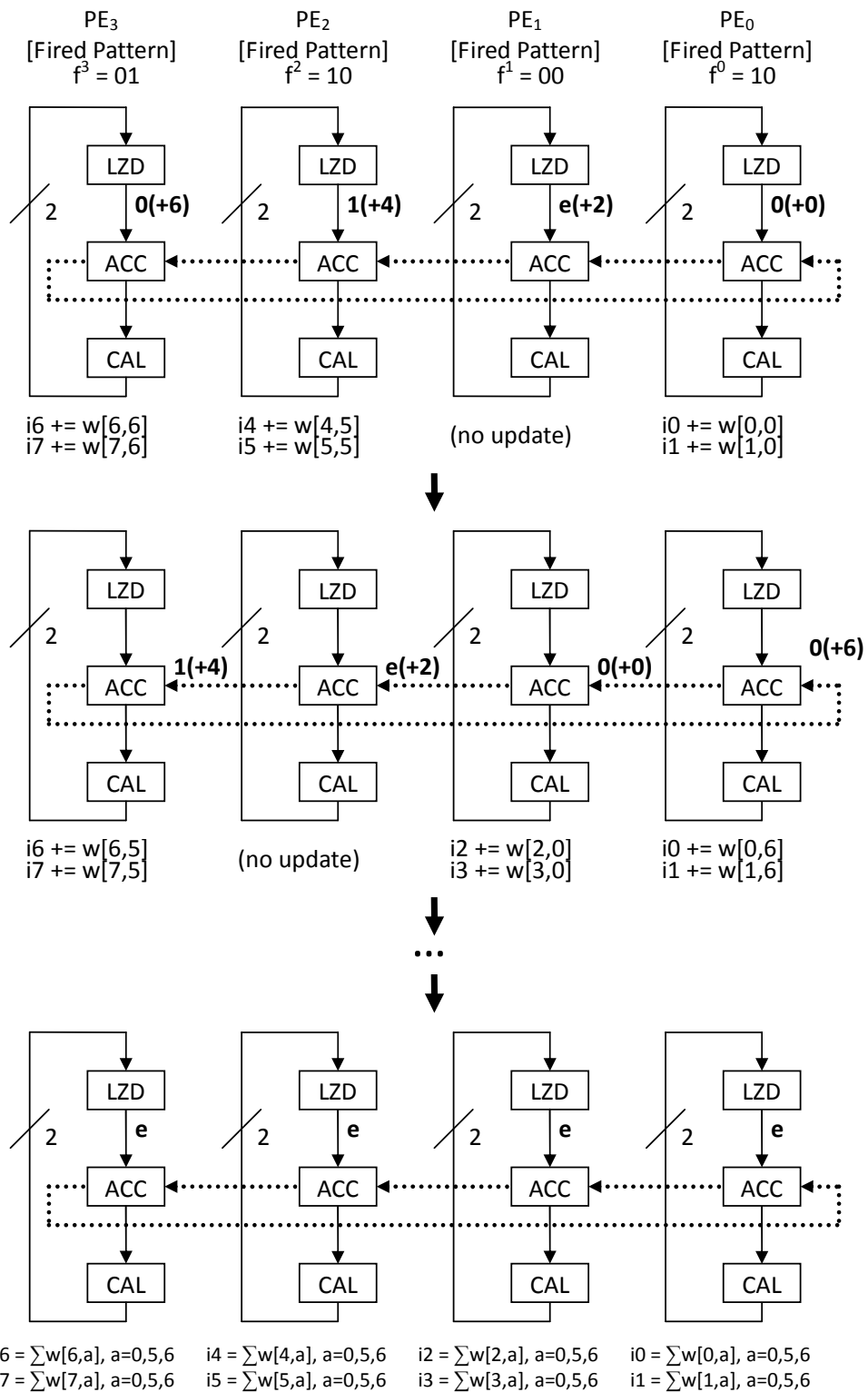


Fig. 2. ACC computation for an 8 neuron simulation. Neurons 0, 5 and 6 have fired and  $I_i$  values are accumulated in the 4 PEs, each one handling two neurons.

in mV. 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 9 bits 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 [7] 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 [8].

### C. Memory Organization

Three different types of data values need to be stored in CAL unit for an implementation of Izhikevich's model. These are the neuron states  $v$  &  $u$ , the neuron parameters  $ab$ ,  $1 - a$ ,  $c$  &  $d$  and synaptic strength  $\mathbf{W}$ .  $ab$  &  $1 - a$  are stored instead of  $a$  &  $b$  since it reduces the delay of the equation 1 of Izhikevich model by 2 levels by evaluating  $u \leq (ab)v + (1-a)u$  instead of  $u \leq a(bv - u) + u$ . This representation is suggested by [4] in their implementation of spiking neural network on FPGA. The synaptic strength dominates the storage area since the required storage area is  $O(N^2)$  whereas the others are  $O(N)$ . For this reason,  $\mathbf{W}$  are stored in dedicated Block RAMs [6] whereas the states and parameters are stored in Distributed RAM [6].

Each neuron has its own table of width  $N$  each storing the synaptic strength  $\mathbf{W}[i, :]$ , where  $N$  is the number of neurons to be simulated. Since the size of Block RAMs is 18Kbit [6], each can hold at most 2000 9-bit synaptic strength values. To simulate a network with  $B < N \leq 1000$  where  $B$  is the number of Block RAMs available on the target device, a single address can store 2 or more synaptic values to increase the memory utilisation.

### D. ACC Unit

During the CAL phase, the spike accumulation is achieved by accumulating one entry of each synaptic strength table in every cycle. Fig. 3 shows the architecture of the ACC unit. The memory tables  $\mathbf{W}[i, :]$  for  $i \in [kC + 1..k(C + 1)]$  are stored in the ACC unit of PE  $k$ . As a result, a total of  $\text{ceil}(C/2) * K$  Block RAMs are required in the full implementation.

The Block RAM receives an address from LOD at 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 address source of the Block RAM. The counter will reset itself whenever it reaches  $K$  to repeat the above procedure. The CAL unit selects the accumulated values through a multiplexer during CAL phase.

### E. CAL Unit Architecture

A pipelined CAL unit produces an output every cycle with a latency of 6 cycles. The datapath is shown in Fig. 4. A counter looping through 0 to  $C-1$  during the CAL phase is

TABLE I  
RESOURCE USAGE

| Resource        | Used (%)    |
|-----------------|-------------|
| Occupied Slices | 13635 (56%) |
| Flip Flops (FF) | 31960 (32%) |
| LUT             | 35229 (36%) |
| DSP             | 128 (100%)  |
| Block RAM       | 208 (98%)   |
| Clock Rate      | 110.47MHz   |

TABLE II  
RESOURCE UTILISATION FOR THE  $K = 32$  &  $C = 25$  CASE.

used to control the input and output sequence of the data. It is connected to the accumulator selection of the ACC unit and the read address of the memory tables. The neuron update pipeline receives the accumulator value, neuron parameters and states and produces a new  $u$  &  $v$  after 6 cycles. The new  $u$  &  $v$  is connected to the write port of the neuron states table. The write address of the table is connected to counter value minus 6 since the output is delayed by 6 cycles.

The implementation of equation 1-3 includes 5 multiplication, in which one of them is a constant-variable multiplication which can be implemented without a DSP block. Hence the unit requires at least 4 multipliers, and the number of DSP blocks available on the device is the limiting factor of the number of PEs  $K$  of the device.

## V. RESULTS

A fully-connected 800-neuron network was implemented on a Xilinx Virtex-5 XC5VLX155T device. It is a medium-size device comparing to other FPGA devices in the Virtex-5 family. The device has 424 18Kbit single port Block RAMs (or 212 36Kbit dual port Block RAMs) and 128 DSP blocks. Hence this device support at most  $128/4 = 32$  PEs and 848 neurons.

Xilinx ISE 9.2i was used to generate the implementation. VHDL generic parameters are used throughout the design so the values of most parameters including  $K$  and  $C$  can be adjusted with ease. Parameters  $K = 32$  &  $C = 25$  and wordlength of 18 bit were used for the implementation. Table II gives a summary of the resource utilisation.

ACC phase requires approximately  $AN$  cycles for the ACC phase and  $C+11$  cycles for the CAL phase, where  $A$  is related to the activity level of the network. Thus a total of  $A * 800 + 36$  cycles are required to simulate a timestep in this implementation.

A simulation with standard level of firing activity of 6.5Hz has been carried out. The results shows that a timestep requires approximately 716.8ns. Using timestep size of 1ms, 716.8us is required to simulate a real time second, which is equal to a speed up of 1395x over real time speed.

The design employed in this work is compared to a pipelined design employed in [4] which also employs Izhikevich's model for small size neural network simulation. The model provides a constant speed up of 148x over real time speed and is not affected by the network's activity level.

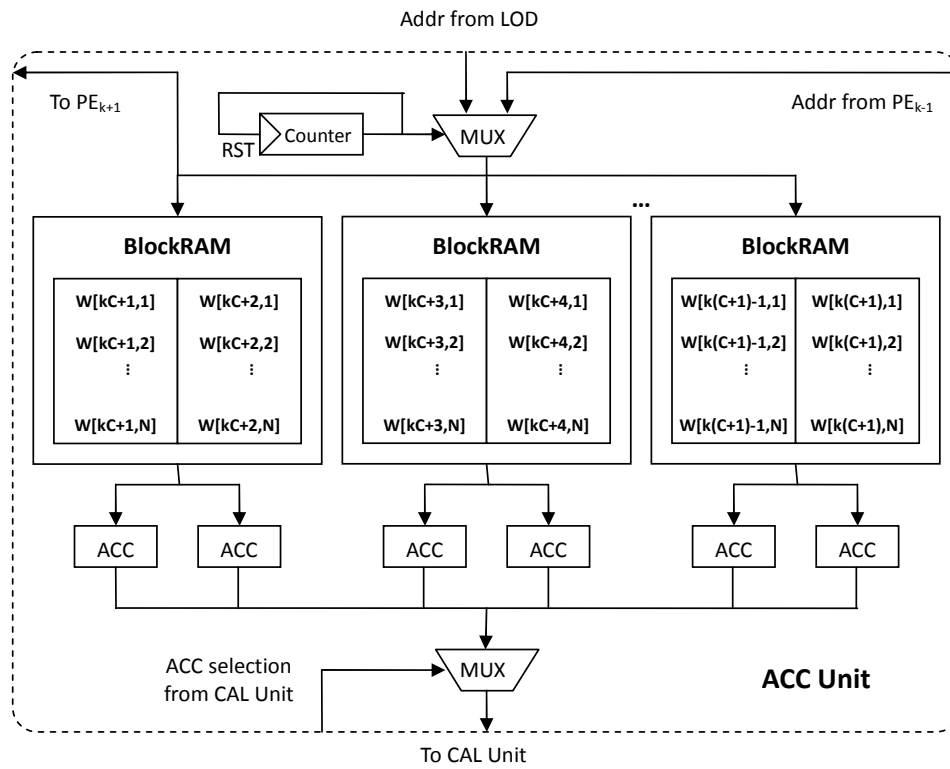


Fig. 3. Architecture of ACC Unit

TABLE III  
DESIGN COMPARISON WITH PIPELINED DESIGN IN [4]

| Our Design                                     | Pipelined Design                                  |
|--|---|
| $AN+C+11$ cycles per time step                 | $N+96$ cycles per time step                       |
| Run time $\propto$ activity level              | Run time is fixed                                 |
| Can be directly cascaded into a larger network | Cannot be directly cascaded into a larger network |
| Slightly complicated architecture              | Slightly simpler architecture                     |
| Require more DSPs                              | Require less DSPs                                 |

## VI. CONCLUSION

\*\*\* stream of thought \*\*\* To mention: (I can write this stuff but perhaps better to do it after you've written the other bits and we can see how well it fits)

- mention Izhikevich's newer paper where he simulates (or pretends to) the entire cortex. There are some small differences between the neuron model used in the earlier paper, which we have followed, and that in the later paper. Our approach certainly applies to the later model, but we have used the earlier one for simplicity [9]
- mention that our architecture allows 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 we build larger and larger scale cortical models, they are not going to come "pre-wired" - and 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.

- comment: I am assuming that it is best for us to just get this idea out there (so if someone else does it first we can insist they cite us!), but if there is a big problem with "scooping" in your field (as there is in mine) it can be better to say little and give the idea only when we've done it. My feeling is the former - just get it out there.

## 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] H. T. Kung and C. E. Leiserson, "Systolic arrays (for vlsi)," in *Proc. SIAM Sparse Matrix Symposium*, 1978, pp. 256–282.
- [3] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1569–1572, 2003.

- [4] 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.
- [5] 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.
- [6] Xilinx Inc., *Virtex-5 FPGA Data Sheet*, 2009. [Online]. Available: [www.xilinx.com](http://www.xilinx.com)
- [7] 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.
- [8] 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.
- [9] 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.

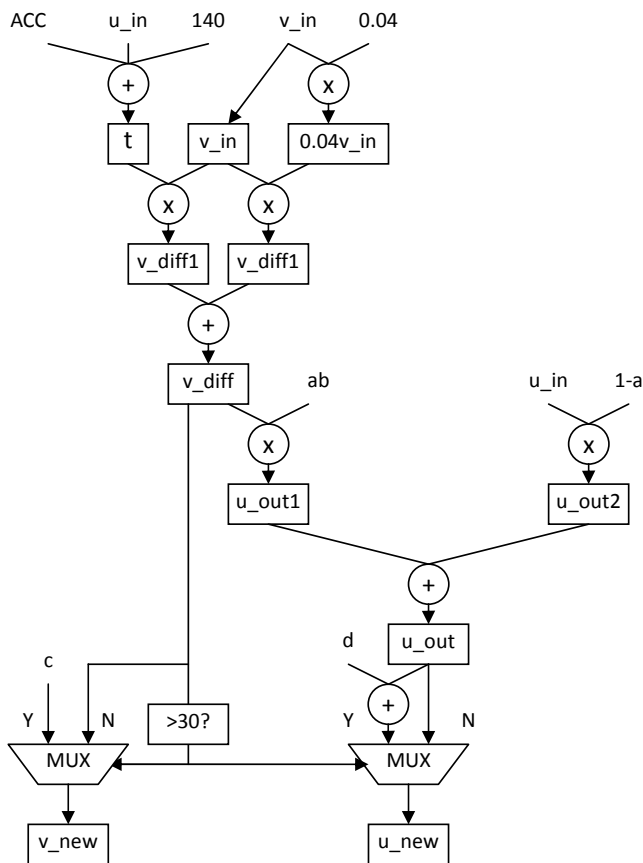
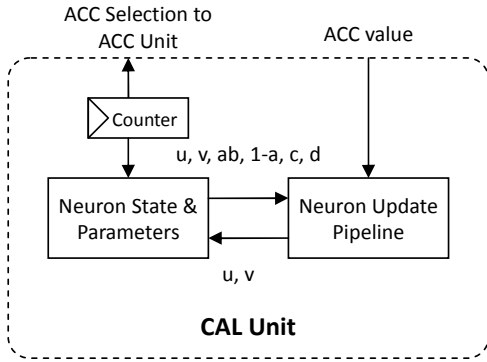


Fig. 4. Datapath of CAL Unit