Hardware Implementation of a Bio-plausible Neuron Model for Evolution and Growth of Spiking Neural Networks on FPGA

Hooman Shayani
Department of Computer Science
UCL, London, UK
Email: h.lastname(at)cs.ucl.ac.uk

Peter J. Bentley
Department of Computer Science
UCL, London, UK
Email: p.lastname(at)cs.ucl.ac.uk

Andrew M. Tyrrell
Department of Electronics
University of York, York, UK
Email: amt(at)ohm.york.ac.uk

***Abstract*— We propose a digital neuron model suitable for evolving and growing heterogeneous spiking neural networks on FPGAs by introducing a novel flexible dendrite architecture and the new PLAQIF (Piecewise-Linear Approximation of Quadratic Integrate and Fire) soma model. A network of 161 neurons and 1610 synapses was simulated, implemented, and verified on a Virtex-5 chip with 4210 times real-time speed with 1 ms resolution. The parametric flexibility of the soma model was shown through a set of experiments.**

I. Introduction

The bewildering complexity of natural organisms, and their remarkable features such as fault-tolerance, immunity, robustness, parallelism, scalability, learning, and adaptation, has made them good sources of inspiration for engineers. Nature-inspired paradigms such as Evolutionary Algorithms (EA), Artificial Neural Networks (ANN), and Artificial Immune Systems (AIS) are all efforts to imitate nature's solutions to create adaptive, intelligent, and secure systems.

However, natural systems simply outperform engineered systems in many aspects. The no-free-lunch theorem [1] implies that, for static and time-dependant problems, there is no algorithm that can perform better than all other algorithms on all problems. This is particularly clear in case of traditional approaches to computation, which have difficulties in solving natural problems like pattern recognition, optimization, and design. They can show an acceptable performance only on a very limited range of problems. For example, a statistical or heuristic object detection algorithm may perform very well for detecting faces in the input images but cannot be used for detecting hands or tools or distorted or incomplete faces. In contrast, natural systems perform very well on a wide set of natural problems. In other words they can adapt to different problems and new situations.

This adaptation of natural systems can show itself in different forms. Scalability can be defined as the ability of a system to grow (*i.e.* to organize and employ more resources) in order to tackle more complicated problems or deal with (proportionally or even super-linearly) larger amount of work. Fault-tolerance can be thought as the ability of a system to cope with resource loss (by graceful degradation) and to reorganize available resources to mitigate, or even recover, the impact of the loss. Robustness could be explained as the quality of a system to be capable of graceful degradation in case of a change (even sudden and/or unpredictable) in its operating environment, including its input. This definition implies generalization over input patterns and robustness to noise.

It seems that all these intrinsic properties of natural systems emerge through intricate interactions of numerous elements on different levels. These elements could be atoms in a folding molecule, large molecules in a living cell, cells in a developing organism, neurons in a learning neural network, individuals in a swarm or replicating systems in an evolving population. All these processes can be imagined as manifestation of the same pattern at different levels, which are perceived on different time-scales. Evolution, development and learning are three major processes of this kind.

Recreating such processes in artificial systems requires a huge amount of resources and computation power. The other option is to select a high level of abstraction and accept the curse of oversimplification, which usually happens in traditional approaches to evolutionary computation [2]. For example, running a bio-plausible developmental evolutionary neural network involves iterative nested cycles of evolution, development, and learning on different time scales. Evolvable hardware may enable us to exploit the computational resources at a lower level, leading to fine-grained system interactions, low-level parallelism, and a biologically more plausible approach compared to traditional evolutionary computing.

This work is a step towards creating adaptable and bio-plausible hardware-based neural networks using evolution, development, and learning processes. Here, we propose a digital neuron model suitable for developmental evolution of heterogeneous recurrent spiking neural networks on FPGAs (Field Programmable Gate Array) aiming at: flexibility, development-friendliness, high simulation and learning speeds, parallelism, and bio-plausibility, while having hardware implementation in mind.

In the next section, the relevant literature is reviewed. In section III we introduce the new digital neuron model and its

978-0-7695-3166-3/08 $25.00 © 2008 IEEE
DOI 10.1109/AHS.2008.13

architecture. The implementation of the digital neurons on a hardware platform is reported in section IV. Experiments on the neuron model and their results are reported in section V and VI. In the last section, these results are analysed and future work is discussed.

II. Background

Many researchers evolved neural networks using different approaches. Yao reviewed some endeavours classifying them into evolving synaptic weights, evolving network architectures, evolving learning algorithms, and their combinations [3]. He concluded that combining evolutionary algorithms and neural networks can lead to significantly better intelligent systems. An evolutionary approach is particularly beneficial in case of Recurrent Neural Networks (RNN) since no systematic and effective approach for designing such networks for a given problem is proposed yet [3], [4].

A. Evolving RNNs

Stanley and Miikkulainen devised an effective way of evolving topology and weights of increasingly complex RNNs called NEAT (Neuro-Evolution of Augmenting Topologies) [5]. It starts from a very simple topology and adds new nodes and connections by mutation, keeping track of the chronological order of innovations to allow meaningful crossovers. NEAT was successfully used in a number of complex control applications [5], [6], [7] and pattern generation [8]. Floreano *et al.* also used evolutionary spiking RNNs for real-time robot control [9].

B. Evolving Reservoirs

Recently, with independent works of Buonomano [10], Maass (Liquid State Machine - LSM [11]), Jaeger (Echo State Network - ESN [12]) and Steil (Back-Propagation Decorrelation - BPDC [13]) a new technique, collectively known as Reservoir Computing (RC) [14], emerged, which is claimed to be capable of processing analogue continuous-time inputs and to mitigate the shortcomings of the RNN learning algorithms. This method is generally based on a recurrent network of (usually) non-linear nodes. This recurrent network, which is called reservoir (AKA liquid, dynamic filter,...) transforms the temporal dynamics of the recent input signals into a high-dimensional representation. This multi-dimensional trajectory can then be used as latent state variables by a simple linear regression/classification or a feed-forward layer (know as readout map or output layer) to extract the salient information from transient states of the dynamic filter and generate stable outputs. The reservoir is traditionally a randomly generated RNN with fixed weights. Only the output layer is trained. Linear nature of the readout map dramatically decreases the computational cost and complexity of the training. Nevertheless, it has been shown that the topology, weights and the other parameters (*e.g.* bias, gain, threshold) of the reservoir elements can change the dynamics of the reservoir and thus affects the performance of the system [4], [15]. Therefore, a randomly generated reservoir is not optimal by definition.

Researchers tried to propose different measures and methods for generating and/or adapting reservoirs for a given problem or problem class [14]. However, there is none or very limited theoretical ground to specify which reservoir is suited for a particular problem due to the non-linearity of the system [4]. Moreover, with only one positive result, in case of intrinsic plasticity [16], the development of unsupervised techniques for effective reservoir adaptation has remained an open research question [14]. Another open question is the effect of the reservoir topology on the performance and dynamics of the system [4]. There is some evidence that hierarchical and structured topologies can significantly increase the performance [14].

Evolutionary algorithms, among other methods, were used for optimizing the performance of the reservoirs [17], [15], which led to some positive results. However, much work is still needed to evolve topologies, structures, and regularization/adaptation/learning algorithms. To make this happen, the evolutionary algorithm must be free to change the size, topology, structure, node properties, and learning algorithm of the RNN. An evolutionary system needs emergent properties such as scalability and modularity to be able to work on different hierarchical levels of reservoirs.

C. Using Developmental Processes

Development as a combination of cell division, differentiation, and growth is believed to be one of the processes that can improve the scalability of evolution while it can also bring other emergent properties like regeneration and fault tolerance to digital systems [18]. Gordon [19], [20] showed that a developmental process can enhance the scalability of evolvable hardware as it uses an implicit mapping from genotype to phenotype. Liu *et al.* [21] proposed a model for developmental evolution of digital systems, which also exhibits transient fault tolerance. Bentley [22] compared human-designed, GP, and developmental evolutionary programs facing damage and showed that the developmental program can exhibit graceful degradation. Federici [23] showed that development can bring regeneration and fault tolerance to spiking neural network robot controllers. Hampton and Adami also proposed a new developmental model for evolution of robust neural networks and reviewed some other developmental neural networks [24]. Roggen *et al.* have a good review and classification of hardware-based developmental systems in [25].

D. Hardware-based POE Systems

Researchers have sought to create systems capable of evolving, developing and learning *in situ* to adapt themselves to a given problem and environment. These are called POE systems as they are aimed to show these capabilities in all three aspects of Phylogeny, Ontogeny and, Epigenesis of an organism. A spiking neural network on the POEtic chip [26] is an example of such systems.

E. FPGA-based POE Spiking Neural Networks

The evolution of directly mapped recurrent spiking neural networks on FPGAs has been tackled by a few researchers

(*e.g.* [25], [27]) using very simplified versions of the Leaky Integrate and Fire model (LIF) [28], [29]. Recently, Schrauwen *et. al* proposed a high-speed spiking neuron model for FPGA implementation based on the LIF model with serial arithmetic and parallel processing of the synapses utilising pipelining in a binary dendrite tree [30].

However, as yet none of these digital neuron models are quite suitable for a developmental model capable of regeneration and dendrite growth on FPGA. They are typically either constricted in terms of number of inputs per neuron or impose constraints on the patterns of connectivity and/or placement on the actual chip mostly due to implementation issues. They also do not allow heterogeneous networks with flexible parametric neurons and learning rules as important bio-plausible features.

III. Digital Neuron Model

To improve the performance of evolution, the developmental digital neuron model should be as flexible as possible, for any constraint may impair evolvability. Evolution must be able to modify everything from network topology and dendrite structures to learning rules, membrane decay constants and other cell parameters and processes. Evolution should also be free to create a suitable neuro-coding technique for each application. Therefore, even the network activity is not guaranteed to be restricted as assumed in event-based simulation of spiking neural networks [30]. Thus, a time-step simulation technique is used here. This model also needs to be relatively fast as running a POE system [26] involves iterative nested cycles of evolution, development and learning. Such a fast parallel spiking neural network on FPGA can also be used for real-time applications. Design objectives of the digital neuron model can be summarised in order of importance as follows:

1) Flexibility and evolvability in terms of:
   - Development-friendliness of the dendrite model
   - Parametric flexibility of the soma model
   - Flexibility of the learning algorithm
   - Flexibility of the neural coding
2) Simulation and learning speed (parallelism)
3) Bio-plausibility
4) Minimization of hardware area and resources on FPGA

A. General Architecture

In the proposed model [31], each digital neuron consists of a set of synapse units and a soma unit connected in a daisy chain architecture shown in figure 1. The pre-synaptic input of each synapse is connected to the axon of the pre-synaptic neuron. This architecture creates a 2-way communication channel and allows the development of different dendrite structures as demonstrated in the example of figure 2. The signal pairs that connect the units form a loop that conveys data packets (comprising a start bit and 16 data bits). The soma unit sends an upstream packet containing the current membrane potential on its upstream output (USO). Synapse units pass upstream packets unchanged but process downstream packets. If a synapse unit receives a pre-synaptic action potential it

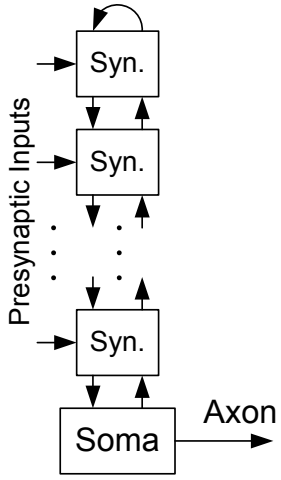

Fig. 1. General architecture of the digital neuron (Syn: synapse unit).

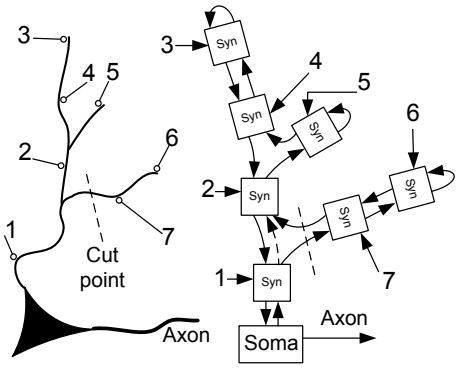

Fig. 2. Example of the dendrite structure and its adaptability (Syn: synapse unit).

adds (or subtracts) its synaptic weight to the first arriving downstream packet. Therefore, the soma unit receives the sum of membrane potential and post-synaptic currents in its downstream input (DSI). After processing this packet, the soma unit sends another packet with the updated membrane potential. Serial arithmetic is used in all the units to create pipelined parallel processing inside each neuron, meaning that neighbouring units process different bits of the same packet at the same time. Using this architecture has a number of collective benefits. First, a 2-way communication channel makes it possible to have a local synaptic plasticity mechanism in each synapse leading to a higher level of parallelism. Most of the bio-plausible unsupervised learning mechanisms like STDP and its variants involve a local learning process in each synapse. Secondly, it minimizes the number of local and global connections, which leads to a significant relaxation of constraints imposed upon the network architecture as limited routing resources is the major constraint in optimal utilization of FPGA functional resources. Each unit needs only a global clock signal to work. Other global signals can also be added for global supervised learning mechanisms. Although other

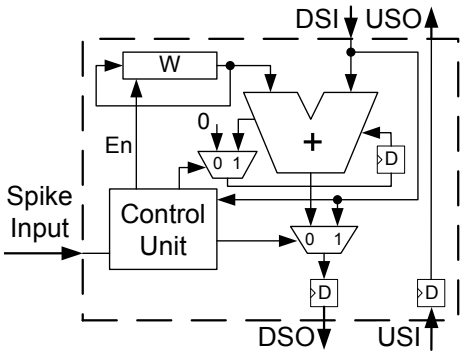

Fig. 3. Internal architecture of the synapse unit.

architectures may bring about less pipeline latency, they need more local and global connections. For instance, a binary tree structure similar to [30] needs about double the number of local connections including the upstream links (excluding the global control signals). Third, it allows the development of dendrite structures similar to biological dendrites. The user is free to trim (add) dendrite sub-trees at any point simply by cutting few connections and bypassing (inserting) the root unit of the sub-tree as shown by the dashed lines in figure 2. This can be implemented in FPGA using multiplexers or other routing resources (The detail is beyond the scope of this paper). This flexibility is vital for a developmental model that needs on-line growth and modification. Fourth, it maintains the regularity of the model by limiting the diversity of the module types (synapse and soma units) and connection types (dendrites, axons) to a biologically plausible bare minimum. This simplifies the place and route or dynamic reconfiguration process if a regular infrastructure of cells and connections (similar to [32]) is used. Finally, it is possible to add other variables to the data packet (*e.g.* the membrane recovery variable in the Izhikevich model [33]).

B. The Synapse Unit

The synapse unit, shown in figure 3, comprises a 1-bit adder (with carry flip-flop), a shift register containing the synaptic weight, two pipeline flip-flops, and a control unit. The upstream input (USI) is simply directed to the upstream output (USO) through a pipeline flip-flop. The control unit disables the adder and weight register when no spike has arrived by redirecting the downstream input (DSI) to the downstream output (DSO) through another pipeline flip-flop. When the control unit detects a spike, it waits for the next packet and resets the carry flip-flop of the adder when it receives the start bit. Then it enables the shift register and the adder until the whole packet is processed. A learning block can be simply inserted into the feedback loop of the weight register in order to realize a local unsupervised learning mechanism like STDP. This learning block can access the current membrane potential and the input. It is also possible to modify the synapse to create a digital DC current input unit by loading the DC current into the weight register.

C. The Soma Unit (PLAQIF Model)

Most of the hardware models are based on the Leaky Integrate and Fire (LIF) [29], [28] or simplified LIF neuron models [25], [27]. However, a Quadratic Integrate and Fire neuron model (QIF) is biologically more plausible compared to the popular LIF model as it can generate action potentials with latencies, has dynamic threshold and resting potentials and it can have two bistable states of tonic spiking and silence [34]. Here, a Piecewise-Linear Approximation of the Quadratic Integrate and Fire (PLAQIF) is proposed as a new soma model. Using this new model has a number of benefits in our context.

While it is relatively inexpensive (in terms of hardware resources) to convert a serial arithmetic implementation of a LIF neuron model into a PLAQIF model (as shown later), PLAQIF model can generate a bio-plausible action potential. This is particularly important as we use the membrane voltage in the learning process.

Moreover, the Behaviour of the model can be specified with a number of parameters (*i.e.* time constants and reset potential). These parameters can be placed in registers and look-up-tables (LUT) to be modified at run-time (*e.g.* by partial dynamic reconfiguration) or can be hard-wired for hardware minimization.

Finally, it is easy to extend this model to a piecewise-linear approximation of Izhikevich model (with a wide range of bio-plausible behaviours *e.g.* bursting, chattering, and resonating [33]) by adding another variable, if hardware budget permits.

The dynamics of the QIF model can be described by a differential equation and reset condition of the form [29]:

$$\begin{aligned} \dot{u} &= a(u-u_r)(u-u_t)+I \\ \text{if } u \geq u_{peak} &\text{ then } u \leftarrow u_{reset} \end{aligned} \tag{1}$$

where u is membrane voltage, a is specifying the time-constant, I is the postsynaptic input current, and the u_r and u_t are nominal resting and threshold voltages respectively, when $I = 0$. Note that in contrast with LIF models, the actual resting and threshold voltages are dynamic and they change with the input current I [34]. Applying first-order Euler method results in an equation of general form:

$$\begin{aligned} u_{k+1} &= u_k + a(u_k-u_r)(u_k-u_t)+I_k \\ \text{if } u_{k+1} \geq u_{peak}, &\text{ then } u_{k+1} \leftarrow u_{reset} \end{aligned} \tag{2}$$

where k is the step number. The PLAQIF model is based on the serial arithmetic implementation of a LIF model, with equation $u_{k+1} = u_k + I - au_k$ (for $a < 1$), with a little modification. The last term of LIF equation can be approximated using two taps:

$$u_{k+1} = u_k + I_k + \underbrace{\left\lfloor \frac{u_k}{P_1} \right\rfloor}_{\text{Tap 1}} + \underbrace{\left\lfloor \frac{u_k}{P_2} \right\rfloor}_{\text{Tap 2}} \tag{3}$$

where $P_i = (-1)^{s_i} \cdot 2^{p_i}$ with p_i and s_i being the parameters of ith tap. Each tap is computed by adding (or subtracting depending on s_i) the shifted version (arithmetic shift right by p_i bits) of the binary representation of u_k.

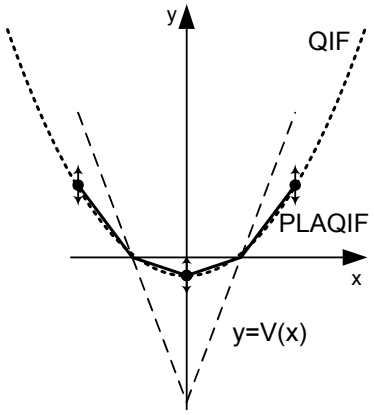

Fig. 4. The PLAQIF model approximates the QIF model (the dotted curve) with a piecewise linear function by modulating the V-shape function $V(x)$. The three control points (with arrows) can be moved up and down by tuning the parameters.

By replacing the sign bit (S) and the most significant bit (MSB) of u_k with the complement of MSB we can produce the piecewise linear function $V(u_k) = |u_k| - 2^{14}$ (assuming a 16-bit representation). This function is shown in figure 4 as the V-shape function. By tapping (modulating) $V(u_k)$ with different parameters ($p_{i,0} \dots p_{i,3}$ and $s_{i,0} \dots s_{i,3}$) for different combinations of S and MSB (positive or negative, small or large values of u_k) we get:

$$u_{k+1} = u_k + I_k + \left\lfloor \frac{V(u_k)}{P_1(u_k)} \right\rfloor + \left\lfloor \frac{V(u_k)}{P_2(u_k)} \right\rfloor \quad (4)$$

$$\text{where } P_i(x) = (-1)^{s_{i,j}} \cdot 2^{p_{i,j}} \ , \quad j = \left\lceil \frac{x}{2^{14}} \right\rceil + 1 \quad (5)$$

It is possible to approximate eq. 2 with eq. 4 by tuning the parameters $p_{i,j}$ and $s_{i,j}$ as shown in figure 4.

The soma unit, showed in figure 5, comprises a 1-bit adder, a 32-bit buffer shift register (holding the partial sums from the last cycle), a 16-bit shift register (holding reset voltage u_{reset}), a lookup-table (LUT, a 8x5 bits RAM, which holds the parameters $p_{i,j}$ and $s_{i,j}$), a control unit (CU, which detects the arriving packet and generates all the control signals *e.g.* Tap, ShiftEn, etc.), and a few multiplexers. The soma unit initiates a data packet thorough USO and waits for a packet on DSI input. At this point, the buffer holds the value u_k in its left half and S and MSB flip-flops hold the sign and most significant bit of u_k. The LUT selects the correct shifted version (according to S and MSB) of u_k through the multiplexer and has its first bit ready on the input of the adder. The first tap starts with receiving a packet. An arriving packet, which contains the value $u_k + I_k$ goes to the other input of the adder. The LUT also selects the add or subtract operation in each tap(s_i). As the operation goes on, the MSB extension block switches the multiplexer to $\overline{\text{MSB}}$ at the right time to generate the value $\left\lfloor \frac{V(u_k)}{P_1(u_k)} \right\rfloor$ on the input of the adder. Therefore, the new value of $u_k + I_k + \left\lfloor \frac{V(u_k)}{P_1(u_k)} \right\rfloor$ shifts into the buffer through a multiplexer. The second tap starts immediately and the value in the left half of the buffer goes to the adder input. The other input of the adder is again $\left\lfloor \frac{V(u_k)}{P_2(u_k)} \right\rfloor$ now generated by selecting the correct shifted version of the u_k from the right half of the buffer. The adder generates the updated value of u (u_{k+1} in eq. 4) at its output, which is shifted into the buffer and is also used to generate a new packet in the upstream output of the soma unit. This value is also used to update the S and MSB flip-flops according to the new value of u_{k+1}. This process continues until the peak detection block detects a transition of S without any change in MSB, which indicates an overflow, and immediately corrects the sign bit of the departing packet, generates a pulse in the axon, and initiates the absolute refractory period. The absolute refractory period, which lasts for a complete membrane update cycle, is like any other cycle except that in the second tap the output of the adder is ignored and contents of the reset voltage shift register is used instead as the new membrane potential u_{k+1}. The membrane update period (*i.e.* latency of the whole pipeline), thus neuron time constants, depend on the number of synapses n ($T = 2n + 18$ clock cycles). This can be compensated by the evolving parameters.

IV. Implementation

The behaviour of the neuron model was verified by VHDL simulation of a single neuron. Random spikes were fed into 16 synapses with different weights using different bio-plausible parameter settings and its membrane potential was monitored and compared to the expected dynamics of equation (4). With efficient use of the 32-bit shift registers in Virtex-5 FPGA, a random small-world network of 161 16-bit neurons with 20 inputs, 20 outputs and, 10 fixed-weight synapses per neuron, was simulated and synthesized for a XC5VLX50T chip using VHDL and Xilinx ISE tools resulting 85% utilization and a maximum clock frequency of 160MHz (*i.e.* pipeline throughput, which allows 4210 times faster than real-time simulation with 1 millisecond resolution). We believe that we can improve some of these figures by low-level design optimization and a cellular floor-planning similar to [32].

A single neuron and a whole network were also implemented and tested with a XC5VLX50T chip on a Xilinx ML505 development platform [35]. Through an informal verification process, timings of the neuron output spikes were checked against the simulation at the maximum clock frequency of 160 MHz using random input pulses with the same bio-plausible parameter settings use in simulation. The hardware behaviour matched the simulated model.

V. Experiments

To show the parametric flexibility of the new PLAQIF soma model and to compare its behaviour and capabilities with those of biological neurons and hardware neuron models, four experiments were carried out. To explore a wide range of behaviours, arbitrary different parameter settings were selected.

In the first experiment we checked if the neuron model is capable of showing both bistable and monostable behaviours of biological neurons. For bistable behaviour the u_{reset} was

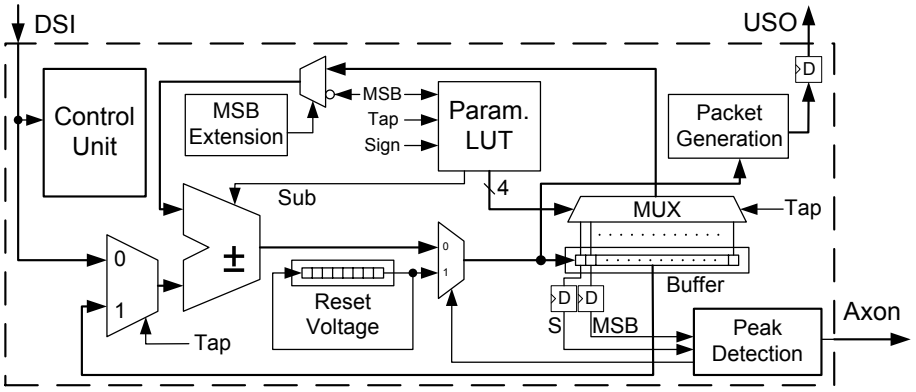

Fig. 5. Internal architecture of the soma unit

set to 17000 and for monostable behavious $u_{reset} = -16384$. The other parameters were set as follows:

$$P_1(x) = \begin{cases} 2^7 & 0 \leq x \\ -2^7 & x < 0 \end{cases}$$

$$P_2(x) = \begin{cases} 2^5 & 0 \leq x \\ -2^5 & x < 0 \end{cases}$$

In the second experiment, to check the effect of changing u_{reset} on the the F-I curve (spiking frequency against input current) of the neuron, the F-I curve was recorded using different values of the parameter u_{reset}, keeping all other parameters fixed as follows:

$$P_1(x) = P_2(x) = \begin{cases} 2^4 & 2^{14} \leq x \\ 2^3 & 0 \leq x < 2^{14} \\ -2^4 & -2^{14} \leq x < 0 \\ -2^3 & x < -2^{14} \end{cases}$$

In the third experiment, the F-I curve was recorded changing the middle control point (in figure 4) keeping all other parameters fixed ($u_{reset} = -16384$ and for two other control points: $P_1(x) = 2^7$ and $P_2(x) = 2^3$ when $|x| \geq 2^{14}$).

In the forth experiment, only u_{reset} was fixed at -16384 and the F-I curves for a few different symmetric settings of $p_{i,j}$ and $s_{i,j}$ (where $P_i(x) = -P_i(-x)$, $i = 1, 2$) were recorded. For comparison, a QIF model of the form:

$$\dot{u} = 0.1u^2 + \frac{I}{8000} - \frac{2174}{8000} \qquad (6)$$
$$\text{if } u \geq 30 \text{ then } u \leftarrow -1$$

was also simulated with the same resolution.

All experiments were carried out using VHDL simulation of a single neuron with 16 synapses with their weights set to $2^0, 2^1, \cdots, 2^{14}, -2^{15}$. Each frequency measurement was made by first setting all the synaptic inputs to zero for 2 membrane update cycles and then fixing the binary representation of the input current on the synaptic inputs, waiting for the first spike in the axon and counting the number of update cycles until the second spike (N). The frequency was then calculated assuming that each update cycle is equal to 1 ms of neuron simulation time ($F = \frac{1000}{N}$).

VI. Results

Figure 6 show traces of the input current, membrane voltage and the axon output of the digital neuron in two different settings of the first experiment. The PLAQIF model is clearly capable of working in both bistable and monostable modes and generating spikes with latencies. This is in contrast with the popular LIF model that works only in the monostable mode and cannot generate spikes with latencies.

Figure 7 shows the results of the second experiment that demonstrates the effect of changing the parameter u_{reset} on the F-I curve of the neuron. The F-I curve clearly shows that the digital neuron is class 1 excitable [29] for $u_{reset} < 0$. The PLAQIF model F-I curve apears as a class 2 excitability [29] for $0 \leq u_{reset} \leq 16384$. This is also an advantage over LIF model. Moreover, changing u_{reset} affects the general slope and curvature of the neuron F-I curve. For positive values of u_{reset}, the minimum spiking frequency and current change as u_{reset} changes.

Figure 8 shows the results of the third experiment. A class 1 excitability is more clear in this figure. It also shows how the slope and curvature of the F-I curve can be fine-tuned by changing the middle control point (in figure 4) parameters. The bold lines show the F-I curve when the middle control point is higher than zero (W shaped function instead of V-shaped or quadratic function for $\dot{u}(u)$). These exotic non-linearities in $\dot{u}(u)$ that do not match contemporary biological neurons can be exploited during evolution.

Results of the last experiment, shown in figure 9, demonstrates diversity of neuron characteristics using different parameter settings without changing u_{reset}. The F-I curve of the QIF model of 6 is shown in bold, which is close to the F-I curve of the PLAQIF model with the parameter settings:

$$P_1(x) = \begin{cases} 2^7 & 0 \leq x \\ -2^7 & x < 0 \end{cases}$$

$$P_2(x) = \begin{cases} 2^3 & 0 \leq x \\ -2^3 & x < 0 \end{cases}$$

It is an acceptable approximation, however the PLAQIF curve does not exactly match the QIF curve due to the piecewise-linear approximation.

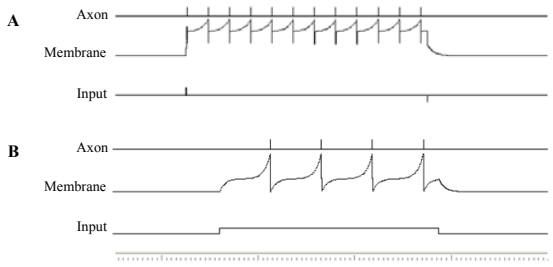

Fig. 6. Traces of the input current, membrane potential, and the axon output of the digital neuron in the first experiment: A) Bistable behaviour ($u_{reset} = 17000$). B) Monostable behaviour ($u_{reset} = -16384$).

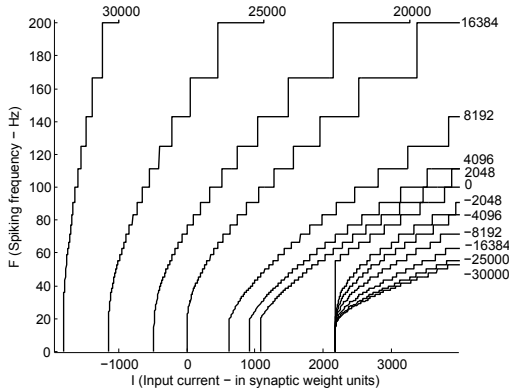

Fig. 7. F-I curve of the digital neuron using different values of u_{reset} showing class 1 and 2 excitability.

The step-wise shape of the curves (particularly in higher frequencies) is due to the 1-millisecond simulation resolution and calculating the frequency based on the number of 1-millisecond update cycles between two successive spikes.

VII. Conclusions

A digital spiking neuron model with a new architecture and a novel soma model based on piecewise-linear approximation of QIF neuron model was proposed and its suitability for evolution and development of heterogeneous neural networks in FPGAs was shown in terms of parametric flexibility of the soma units, adaptability of the dendrite structures, and model simulation speed. It was also shown how a local learning unit can be added to each synapse to improve parallelism. Although the replication of the synapse control units increases the hardware area, it brings about the adaptability of the dendrite structures and may also adds to the fault-tolerance of the network. Compared to existing hardware-based models (*e.g.* [30], [36], [25], [26]), the new digital neuron model has many advantages:

1) Due to the non-linearity of the new PLAQIF soma model (based on QIF model) it:
   - can generate bio-plausible spikes with latencies;
   - can behave in both monostable and bistable modes;
   - can show both class 1 and class 2 neuron excitability;
   - has dynamic resting and threshold potentials.
2) With a wider set of parameters (4 time constants and a reset voltage), it has more useful flexibility to be exploited by evolution. It is possible to generate exotic non-linearities with PLAQIF that even QIF model can not produce.
3) It can form unlimited number of synapses and flexible dendrite branches without any connectivity or placement constraint or need for global control signals to be routed allowing online development and dendrite growth.
4) It allows unsupervised learning mechanisms to be implemented (or, in future designs, even evolved) locally in each synapse. While this is a bio-plausible approach to learning, it also creates low-level parallelism in the learning process leading to higher speeds.

Compared to [30], this model is 43% faster (having 10 synapses per neuron), but needs more hardware resources. This

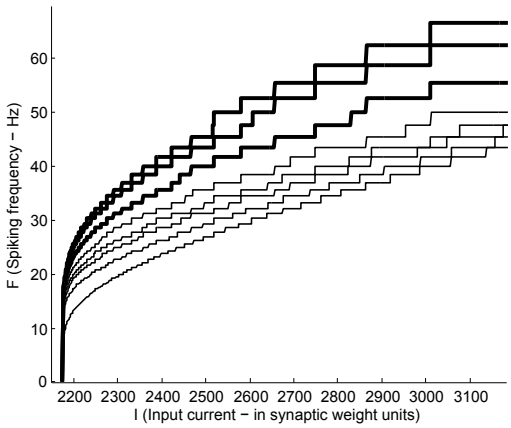

Fig. 8. The effect of moving the middle control point (of figure 4) on the F-I curve of the digital neuron. The bold lines are results of the middle point higher than zero.

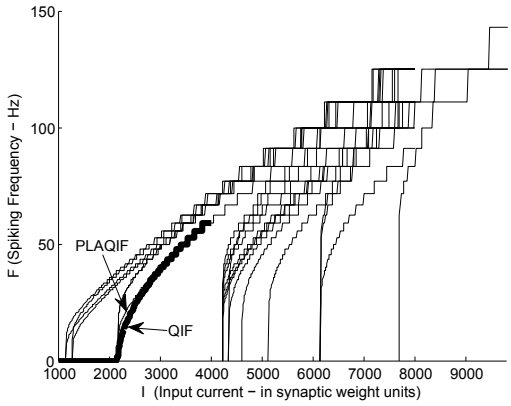

Fig. 9. F-I curve of the digital neuron using different parameter settings for $p_{i,j}$ and $s_{i,j}$ keeping $u_{reset} = -16384$ along with the F-I curve of the QIF model of eq. 6 superimposed in bold.

speed advantage is reduced for higher number of synapses per neuron as the update cycle of this architecture is of order $\mathcal{O}(n)$ compared to $\mathcal{O}(\log_2 n)$ in [30]. However, it is not very useful to compare these two designs due to differences in technologies and priorities of the design objectives (flexibility and adaptability instead of speed and hardware minimization). Clearly, using the new daisy-chain architecture and the replication of the synapse control units increased the hardware resources and slightly reduced the speed but also contributed to the adaptability of the dendrite structure and the real possibility of introducing meaningful development into the process.

In the next step of this study, it is intended to design a versatile cellular architecture for development and dendrite growth of digital neurons of this type on Virtex-5 FPGA. Adding a parametric evolvable learning (synaptic plasticity) block to the synapse unit and evolving heterogeneous neural networks on Virtex-5 FPGA using dynamic partial reconfiguration is also planned for future.

REFERENCES

[1] D. H. Wolpert and W. G. Macready, "No Free Lunch Theorems for Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, Apr. 1997.

[2] J. Timmis, M. Amos, W. Banzhaf, and A. Tyrrell, ""Going back to our roots": second generation biocomputing," 2005. [Online]. Available: http://www.citebase.org/cgi-bin/citations?id=oai:arXiv.org:cs/0512071

[3] X. Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.

[4] H. Jaeger, "Reservoir riddles: suggestions for echo state network research," in *Neural Networks, 2005. IJCNN '05. Proceedings. 2005 IEEE International Joint Conference on*, vol. 3, 31 July-4 Aug. 2005, pp. 1460–1462vol.3.

[5] K. O. Stanley and R. Miikkulainen, "Evolving Neural Networks through Augmenting Topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[6] K. Stanley and R. Miikkulainen, "Competitive coevolution through evolutionary complexification," *Journal of Artificial Intelligence Research*, vol. 21, no. 1, pp. 63–100, 2004.

[7] K. Stanley, "Real-time neuroevolution in the NERO video game," *IEEE transactions on evolutionary computation*, vol. 9, no. 6, pp. 653–, 2005.

[8] ——, "Compositional pattern producing networks: A novel abstraction of development," *Genetic programming and evolvable machines*, vol. 8, no. 2, pp. 131–, 2007.

[9] D. Floreano, Y. Epars, J. Zufferey, and C. Mattiussi, "Evolution of Spiking Neural Circuits in Autonomous Mobile Robots," *International Journal of Intelligent Systems*, vol. 20, p. 100, 2005.

[10] D. V. Buonomano and M. M. Merzenich, "Temporal Information Transformed into a Spatial Code by a Neural Network with Realistic Properties," *Science*, vol. 267, no. 5200, pp. 1028–1030, Feb. 1995.

[11] Maass, Natschlager, and Markram, "Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations," *NEURCOMP: Neural Computation*, vol. 14, pp. 2531–2560, 2002.

[12] H. Jaeger and H. Haas, "Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication," *Science*, vol. 304, no. 5667, pp. 78–80, 2004. [Online]. Available: http://www.sciencemag.org/cgi/content/abstract/304/5667/78

[13] J. Steil, "Backpropagation-decorrelation: online recurrent learning with O(N) complexity," in *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, vol. 2, 25-29 July 2004, pp. 843–848vol.2.

[14] B. Schrauwen, D. Verstraeten, and J. Van Campenhout, "An overview of reservoir computing: theory, applications and implementations," in *Proceedings of the 15th European Symposium on Artificial Neural Networks*, 2007, p. 471482.

[15] M. Lukosevicius and H. Jaeger, "Overview of Reservoir Recipes," School of Engineering and Science, Jacobs University, Bremen, Germany, Tech. Rep. 11, July 2007.

[16] J. Steil, "Online reservoir adaptation by intrinsic plasticity for backpropagation–decorrelation and echo state learning," *Neural Networks*, vol. 20, no. 3, pp. 353–364, 2007.

[17] T. van der Zant, V. Becanovíc, K. Ishii, H. Kobialka, and P. Ploger, "Finding good echo state networks to control an underwater robot using evolutionary computations," in *Proceedings of the 5th IFAC symposium on Intelligent Autonomous Vehicles (IAV04)*, 2004.

[18] J. Miller, "Evolving Developmental Programs for Adaptation, Morphogenesis, and Self-Repair," in *Advances in Artificial Life. 7th European Conference on Artificial Life*, ser. Lecture Notes in Artificial Intelligence, W. Banzhaf, T. Christaller, P. Dittrich, J. T. Kim, and J. Ziegler, Eds. Dortmund, Germany: Springer, 2003, pp. 256–265.

[19] T. Gordon and P. Bentley, "Development brings scalability to hardware evolution," in *Evolvable Hardware, 2005. Proceedings. 2005 NASA/DoD Conference on*, 2005, pp. 272–279.

[20] ——, "Bias and scalability in evolutionary development," in *Proceedings of the 2005 conference on Genetic and evolutionary computation*. ACM Press New York, NY, USA, 2005, pp. 83–90.

[21] H. Liu, J. Miller, and A. Tyrrell, "Intrinsic evolvable hardware implementation of a robust biological development model for digital systems," in *Evolvable Hardware, 2005. Proceedings. 2005 NASA/DoD Conference on*, 2005, pp. 87–92. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/EH.2005.32

[22] P. Bentley, "Investigations Into Graceful Degradation of Evolutionary Developmental Software," *Natural Computing*, vol. 4, no. 4, pp. 417–437, 2005. [Online]. Available: http://dx.doi.org/10.1007/s11047-005-3666-7

[23] D. Federici, "A regenerating spiking neural network," *Neural Networks*, vol. 18, no. 5-6, pp. 746–754, 2005. [Online]. Available: http://dx.doi.org/10.1016/j.neunet.2005.06.006

[24] A. N. Hampton and C. Adami, "Evolution of Robust Developmental Neural Networks," in *Artificial Life IX: Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems*, 2004. [Online]. Available: http://www.citebase.org/cgi-bin/citations?id=oai:arXiv.org:nlin/0405011

[25] D. Roggen, D. Federici, and D. Floreano, "Evolutionary morphogenesis for multi-cellular systems," *Genetic Programming and Evolvable Machines*, vol. 8, no. 1, pp. 61–96, 2007.

[26] J. Moreno, Y. Thoma, E. Sanchez, J. Eriksson, J. Iglesias, and A. Villa, "The POEtic Electronic Tissue and Its Role in the Emulation of Large-Scale Biologically Inspired Spiking Neural Networks Models," *Complexus*, vol. 3, no. 1-3, pp. 32–47, 2006. [Online]. Available: http://www.karger.com/DOI/10.1159/000094186

[27] A. Upegui, C. A. Pena-Reyes, and E. Sanchez, "An FPGA platform for on-line topology exploration of spiking neural networks," *Microprocessors and Microsystems*, vol. 29, no. 5, pp. 211–223, Jun. 2005.

[28] W. Gerstner and W. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.

[29] E. M. Izhikevich, *Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting (Computational Neuroscience)*. The MIT Press, November 2007. [Online]. Available: http://www.amazon.fr/exec/obidos/ASIN/0262090430/citeulike04-21

[30] B. Schrauwen and J. van Campenhout, "Parallel hardware implementation of a broad class of spiking neurons using serial arithmetic," in *Proceedings of ESANN*, 2006, pp. 623–628.

[31] H. Shayani, P. Bentley, and A. Tyrrell, "An FPGA-based Model Suitable for Evolution and Development of Spiking Neural Networks," in *Proc of 16th European Symposium on Artificial Neural Networks, Advances in Computational Intelligence and Learning*, 2008.

[32] A. Upegui and E. Sanchez, "Evolving Hardware with Self-reconfigurable Connectivity in Xilinx FPGAs," in *Adaptive Hardware and Systems, 2006. AHS 2006. First NASA/ESA Conference on*, 2006, pp. 153–162.

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

[34] ——, "Which model to use for cortical spiking neurons?" *Neural Networks, IEEE Transactions on*, vol. 15, no. 5, pp. 1063–1070, Sept. 2004.

[35] *ML505/ML506 Evaluation Platform User Guide*, Xilinx, Oct 2007.

[36] D. Roggen, S. Hofmann, Y. Thoma, and D. Floreano, "Hardware spiking neural network with run-time reconfigurable connectivity in an autonomous robot," in *Evolvable Hardware, 2003. Proceedings. NASA/DoD Conference on*, 2003, pp. 189–198.