A Step Forward to Map Fully Parallel Energy Efficient Cortical Columns on Field Programmable Gate Arrays (FPGAs)

Arfan Ghani, Chan H. See, Syed M. Usman Ali
Engineering, Sports and Science (ESS) Academic Group
University of Bolton, Bolton, BL3 5AB, UK
Email: {a.ghani, c.see}@bolton.ac.uk

Abstract: This paper presents energy and area-efficient hardware architectures to map fully parallel cortical columns on reconfigurable platform – Field Programmable Gate Arrays (FPGAs). An area-efficient architecture is proposed at the system level and benchmarked with a speech recognition application. Due to the spatio-temporal nature of spiking neurons it is more suitable to map such architectures on FPGAs where signals can be represented in binary form and communication can be performed through the use of spikes. The viability of implementing multiple recurrent neural reservoirs is demonstrated with a novel multiplier-less reconfigurable architectures and a design strategy is devised for its implementation.

Keywords: *reservoir computing, recurrent neural networks, hardware/software (HW/SW) co-design, reconfigurable computing, FPGAs, cortical columns, neural signal processing*

I. INTRODUCTION

The idea of reservoir computing was initially introduced by Maass [1] and Jaeger [2]. In their works in [1-2], the network activity is regarded as 'reservoir' where a memory-less readout device was used and trained to classify information from an untrained recurrent neural reservoir. Jaeger used analogue sigmoidal neurons as network units and called the model Echo State Network (ESN) [1], while Maass called it Liquid State Machine (LSM) and focused on networks of spiking neurons [2]. Both ESNs and LSMs are generally called reservoir computing (RC) [3]. These RC systems have been applied in a broad range of applications [2. 4-7], i.e. speech recognition, human action recognition and object tracking. There have been several studies in the past to investigate the paradigm of reservoir computing [3, 8-9] but none of them provide any guidelines as how to implement and analyse a stable reservoir on hardware/software (HW/SW) platforms. In order to address this deficiency, authors in [10-11] demonstrated the viability of implementing neural reservoirs on software platforms. The main focus of this research was to investigate and analyse the impact of input connectivity and to elaborate the parameters that affect the stability of neural reservoirs. Software implementation of small scale reservoirs is not a serious bottleneck, however to exploit the inherent parallelism of cortical columns, hardware implementations are essential. Implementing neural based applications on programmable hardware is challenging because the maximum size of a network that can be implemented on a target FPGA is restricted by the logic and arithmetic operators available on a single device. Therefore, a HW/SW co-design strategy has to be devised for implementation of neuro inspired systems on reconfigurable platforms. A specific bottleneck in implementing large scale artificial neurons on reconfigurable platform is the limited number of embedded multipliers available on a single device. The number of multi pliers grows as the square of the number of neurons. A fully connected two layer network of size 10 neurons will require 100 multipliers and if the network size is increased to 100 neurons, it will require 10,000 embedded multipliers [12-14].

This paper is a continuation of the work published in [12] where authors' outlined a framework for possible implementation on reconfigurable platforms. It exploits previously published techniques, namely area efficient multiplier-less architecture, which overcomes the burden of multipliers required for synaptic multiplications [13-14]. In order to investigate the viability of implementing reservoir computing paradigm on HW/SW platform, this work presents area efficient spiking neurons architectures. These architectures are targeted for large scale implementation of neuro inspired cortical columns for computational related tasks such as sensory fusion. The presented architectures are used as the basic building blocks for fully parallel implementation of multiple cortical columns on FPGAs. The hardware architectures implemented are inspired by ‘microcircuits’ which plays a fundamental role in cortical computation [1]. The main purpose of these so called microcircuits is to read out information and communicate with the neighboring microcircuits connected in a columnar fashion. One of the limitations in implementing large scale spiking neural networks on HW/SW platforms is the limited size of the network, its scalability and weight storage for online training. Reservoir computing alleviates the burden of training at the network level where only the readout neurons are used for classification. The proposed architecture fully exploits the scalability and reconfigurability of FPGAs at the network level, where the focus is on three main areas: pre-processing, post-processing and reconfigurable neural reservoir. Pre and post processing is performed in software and fully parallel recurrent neural reservoir or microcircuit is implemented on FPGA hardware. To evaluate the reservoir dynamics, it is tested and benchmarked with an example of isolated spoken digit recognition – Texas Instruments 46-Word (TI46) [15].

The organization of this paper is as following. In Section II, the methodology and experimental details will be elucidated. Section III discusses area-efficient architectures for reservoir implementation on reconfigurable hardware and section IV demonstrates a HW/SW co-design strategy benchmarked with a speech recognition application. Section V concludes the paper.

II. METHODOLOGY

A. Pre-processing

Pre-processing of speech signals is an important step to develop an efficient and robust digit recognition system. It is very important that the silence portion of the speech signal is segregated from the voiced region. This so called silence removal stage is very important to reduce the computational complexity and improve the processing time. A significant amount of data processing could be minimized by accurately detecting three different parts of speech signals (voices, un-voiced and silence) as depicted in Fig. 1.

In Fig. 1, an end-point detection technique is used where signal energy is calculated and a threshold value is determined. The threshold is compared to the standard deviation of the signal power. A sampling rate of 12 KHz is used for a spoken digit '5' for duration of 0.69 seconds (8260 samples) where the total silence time is 0.37 seconds (4440 samples). The actual signal time can be calculated by subtracting the silence time from the total signal time. The actual signal time is 0.32 seconds or 3840 samples. The signal preprocessing time can be improved to 53 % (0.37/0.69 * 100 = 53%).

B. Feature extraction

Feature extraction is an important step to collect data that can be considered as information after applying an appropriate speech coding technique. There are several techniques that could be used for feature selection and a detailed comparison is provided in [3]. In the proposed method, an approach is adopted where a temporal based LPC (Linear Predictive Coding) technique is used for encoding speech signals [16]. LPC is the most powerful speech analysis technique that uses Levinson-Durbin recursive algorithm to accomplish the task [17]. Most speech processing algorithms analyses speech signals frame by frame with a fixed frame rate. It is computationally expensive and not feasible to process all frames in the signal. It also leads to some problems because due to the various signal lengths the total numbers of frames could be different. For this experiment, a total of four frames were selected for each spoken digit in linear distance from the start and end point of the signal, 7 coefficients per time frame over four frames and hence total 28 features per sample were processed. The LPC coefficients from each spoken digit were used as input vectors for testing the baseline feed forward and the reservoir based networks. In reservoir based network, LPC co-efficient from each spoken digit were used to perturb the neural reservoir and membrane states were recorded and used as input vectors for training and testing the backend classifier for isolated spoken digit recognition.

An LPC method can mathematically be written as:

$$\tilde{x}(n) = a_1 x(n-1) + a_2 x(n-2) + ... + a_i x(n-i) \tag{1}$$

In equation 1, $\tilde{x}(n)$ is the predicted signal value, *x(n-i)* the previous observed value, a_i the predictor coefficient where $i = 1 \cdots p$. An error function (MSE) can be calculated as under:

$$E_i = \frac{1}{n} \sum_{j=1}^{n} (x_{(ij)} - \tilde{x}_j)^2 \tag{2}$$

Where $x_{(ij)}$ is the predicted value by the individual sample *i* for target value *j* (out of *n* samples); and $\tilde{x}_j$ is the target value for sample *j*.

$$E(i) = x(i) - \tilde{x}(i) \tag{3}$$

In equation 3, *E(i)* is the calculated error, *x(i)* is the true signal value and *(i)* is the target value. A sampling frequency of 12 KHz was used and an end point detection technique applied for noise removal. A hamming window was used where frames were overlapped and sampled at 50 Hz with each frame size was fixed at 30 *ms*. The rationale behind reservoir computing is to overcome the computational burden of recurrent neural network training. In the paradigm of reservoir computing, the partial response of a recurrent reservoir is observed from outside by any suitable classification algorithm such as back propagation. It is much easier and more computationally efficient to train the output layer only or so called ‘readout’ neurons, instead of training the complete network of recurrent neurons.

C. Reservoir Dynamics

There are two important characteristics of a stable reservoir namely separation and approximation [1]. These two properties are important for a readout network to classify input data. If a reservoir is not capable to differentiate two separate inputs then the readout will not be able to classify the input information. In this experiment, a reservoir was generated in stochastic fashion as stated in a previously published work of the authors [10].

There are parameters which play an important role in stable reservoir dynamics such as type of neurons used in the reservoir, size of the reservoir and their connectivity. An overall classification accuracy depends on factors such as input feature vector used to perturb the neural reservoir. In author's previous work, several experiments were conducted to observe the internal dynamics of the neural reservoir; details are reported in [10].

A reservoir was constructed with mathematical model of neurons as described in equation 4.

$$\tau_m \frac{dV_m}{dt} = -(V_m - V_{resting}) + R_m(I_{syn}(t) + I_{noise}) \tag{4}$$

In equation 4, τ_m is the time constant of membrane, V_m is the membrane voltage, $V_{resting}$ is the membrane resting potential (which is set to 0 V), I_{syn}(t) is the synaptic input current, I_{noise} is a Gaussian random noise. The membrane potential is initialized with a value of (0.0135 V) and membrane threshold, V_{th} is set to 0.015 V. An output spike is fired if the membrane voltage V_m exceeds a threshold voltage, V_{th}. Once an output spike is fired, the membrane potential resets itself to the values of V_{reset}. Our selection of parameters in these experiments is based on the data obtained from Henry Markram's Lab in Lausanne [18].

Information processing in artificial representation of cortical neurons primarily depends on two issues: 1) what model describes spiking dynamics of each neuron and 2) how the neurons are connected. There have been several studies to investigate the connectivity of cortical neurons and further details are provided in [18-22].

D. *Backend Processing*

In order to investigate the classification accuracy, total dataset was divided into training and testing. The dataset comprised of 200 samples in total for digits (0- 9). Each digit was spoken by five different speakers with 4 utterances by each speaker. In total 28 features were used for each sample with Linear Predictive Coding. To evaluate an overall performance of the readout classifier different training sets and hidden layer neurons were investigated. The best performance achieved was limited to 89% for test data sets. In order to compare with the MATLAB based MLP benchmark, a gradient descent with adaptation training algorithm was used where the goal was set to 0.01 (see table 1). The numbers of input to the MLP classifier were equal to the total number of features, different numbers of hidden neurons are shown in Table 1 and the numbers of output neurons were equal to the total number of classes, which are 10 for this experiment.

E. Simulation procedure

The inputs can be fed into the reservoir in two different ways: analogue currents and spike trains. In these experiments, spike trains were used and fed into the reservoir as synaptic currents, as described in Fig.2. There is a substantial evidence that biologically plausible neurons communicate through spike trains [23-24]. To investigate this, the input analogue values were converted into Poisson spike trains and processed for reservoir based classification. A Poisson process can be characterized as an interval process with exponential distribution and can be expressed as:

$$p(t) = \lambda \exp(-\lambda t) \tag{5}$$

In equation 5, *p(t)* is the probability density function of an exponential distribution, the parameter λ is the parameter of the distribution and *t* is the interspike interval. Spike trains were generated where interspike intervals were randomly drawn from an exponential distribution. The recurrent neural reservoir was perturbed by input spike trains and once the reservoir was perturbed with each input digit (encoded into spike trains) and states were recorded. The inputs were used one after another and reservoir states were recorded for each sample separately. The reservoir states were sampled in a linear scale from 0 to 1 at the time step of 0.25 s and in total five states were used for each sample. According to the theory of reservoir computing, a simple readout will suffice to classify inputs with the partial information extracted through a stable reservoir. All these sampled states were collected and used for an offline readout (MLP classifier) training and testing. For a reservoir of size 8 neurons, in total five states were extracted for each sample (each state had 8 membrane potentials) and in total 40 data points (8x5) were used as one training sample for the readout. The readout network was trained with the training samples and tested with the test data set. The MLP classifier consisted of a single hidden layer and 10 output neurons. The same procedure is used for reservoir size of 15 and 27 neurons. In order to quantify the classification accuracy, the standard supervised MATLAB algorithms were investigated while different reservoir sizes were used to investigate the separation property.

The total number of input neurons for feed forward readout network depends on the size of the reservoir. In total 40 inputs are required for a reservoir size of 8 neurons (8x5), 75 for a reservoir size of 15 neurons (15x5) and 135 for a reservoir size of 27 neurons (27x5).

III. RECONFIGURABLE ARCHITECTURE FOR RESERVOIR IMPLEMENTATION

In order to exploit the inherent parallelism of cortical neurons, optimised hardware architectures were developed for FPGA implementation. In a network of spiking neurons, each input neuron receives signals from other neurons with different synaptic strengths at different times. A single neuron is further connected with other neurons in the network through synaptic clusters. These clusters are shared amongst neurons in a network and the membrane dynamics and their corresponding spike firing times are affected by the synaptic efficacy of these clusters, as depicted in Fig.3.

LIF (Leaky Integrate-and-Fire) neuron model was chosen for the implementation of the neural reservoir and the mechanism of synaptic integration was modelled by the following equation:

$$V_s(t) = \sum_{1}^{N_s} w_i x_i(t) \tag{6}$$

Where V_s is the sum of incoming synaptic potentials to the membrane, N_s is the number of synapses, w_i is the synaptic efficacy and x_i are the incoming binary spikes 0 or 1 at time t. These synaptic potentials were accumulated in a membrane and when the total synaptic potential exceeded a certain threshold, an output spike was generated.

The neuron firing dynamics were modelled with the following equation:

$$V_m(t) = \begin{cases} V_{reset} \rightarrow V_m(t) > v_{th} \\ V_m(t-1) + V_s(t) - V_{leakage}(t) \rightarrow otherwise \end{cases} \tag{7}$$

In equation 7, V_m is the membrane potential, V_{reset} is the reset potential, V_m *(t-1)* is the membrane potential at the previous time step, V_s is the sum of synaptic potential and $V_{leakage}$ is the exponentially decreasing leakage voltage with time constant τ. The membrane voltage V_m *(t)* will be at V_{reset} if V_s *(t)* > V_{th}, otherwise the membrane potential will be equivalent to the second term in equation 7.

The time required to simulate a reservoir depends on its size and the node type used to construct the reservoir. Simulating small networks on sequential machines may not be critical but for large scale networks it becomes a significant overhead. The simulation time increases many folds if larger reservoirs are to be simulated. In order to evaluate the simulation time requirement on sequential machines, different reservoirs were simulated on Intel Pentium P4 (3.20 GHz speed and I GB RAM). As shown in Fig. 4 that the simulation time increases almost at the order of 2.5 which makes it impractical to simulate large scale reservoirs on software platforms.

Hardware implementation of spike based neurons is advantageous because these neurons communicate through short pulses (spikes) and the information is conveyed through exact timing of these pulses where the shape of the spike has no relevance to the information [21]. For an area-efficient implementation of a neural reservoir on reconfigurable hardware, it is necessary that the use of area hungry operators such as multipliers are minimised or completely avoided [13- 14]. In traditional modelling of synapses, inputs are multiplied with fixed weights and due to this multiplication the number of multipliers increases with an increased number of synapses. This is a serious bottleneck for an efficient implementation of medium to large scale networks on a single FPGA device. In the proposed design, special emphasis is given to the minimisation of the number of embedded multipliers required for the implementation of synapses.

The architecture was split into two sub structures: synapse and membrane. For synapse modelling, a stochastic strategy is proposed where inputs were encoded in spike trains and spike counters were used to model synaptic strengths. The incoming spikes were counted and weighted through a fixed weight value. An output value of ‘1’ is generated through a simple logic AND function when both inputs were high (see Fig. 5).

As shown in Fig. 5, each neuron has multiple synapses where input pulses were counted and weighted through a fixed weight value. The synapse function implemented in the proposed architecture is a simple logic function of two inputs (incoming spike trains and fixed weight values). The fixed weight values were stored in the registers and random values were generated through a linear feedback shift register (LFSR). The fixed weight values were compared with the randomly generated values and if the generated value equals the fixed weight value and the number of incoming pulses were equal to the value of pulse counter then an output spike is generated, if not, no pulse is generated. The pulses generated accumulate and hence contribute to the overall membrane potential. This procedure is repeated during the course of the full presentation of the input spike trains. The random weight generation on FPGA was performed with the Xilinx System Generator’s (XSG) [25]“LFSR” block. This block supports both the Fibonacci and Galois structures. A Fibonacci structure was chosen by using XOR gate at the beginning of the register chain that XORs the outputs from some of the registers going into the first register. The LFSR output was set up to start at a specified initial seed value and step through a repeatable sequence of states determined by the LFSR Fibonacci structure, XOR gate and initial seed. The random weight generator block (LFSR) generates new weights at each time step in the range of ± 0.5. A total of 6 bits were used for their implementation and due to this simplified yet area efficient technique, the multipliers were completely avoided and synapse multiplication was modelled with a logic function of two variables W (fixed weights) and I (incoming spikes). The weights were represented with a fixed point representation of 4 bits in a Fix_4_3 format.

The synaptic values of '1' were scaled down through shift right operations. This scaling is important for practical reasons so that enough time is given to the membrane potential to accumulate synaptic inputs and once the total membrane potential exceeded a threshold value, an output spike was generated and connected with other neurons in the network. The threshold voltage was set to 0.15 V and reset voltage to 1 mV. In the absence of spikes, the membrane potential decays exponentially to the reset voltage based on the programmable value of the decay constant. A decay constant value of -0.11 was used in these simulations. The parameters selection is empirical and based on the spiking behaviour of LIF neuron model.

The second half of the architecture is implemented as a neural membrane as shown in Fig. 6, where synaptic currents (synapse accumulation unit) are accumulated in the membrane (accumulator) and an output spike is generated when the total membrane potential exceeded a programmable threshold V_{th}. The threshold is modelled with a comparator block and after spike generation, the membrane potential was set to a 'reset' value through a register. The membrane of the neuron is implemented as an 18 bit accumulator with 12 bit binary points. It should be noted that a fixed point precision "Fix_18_12" was used for synaptic accumulation in the neural membrane by taking into account the minimum area utilization on FPGA.

A programmable threshold value of 0.15 V is used and after spike generation the accumulator was reset to the value of 1 mV. In the absence of input spikes, the membrane potential decays exponentially to the reset voltage and starts integrating after arrival of new incoming spikes. The exponential decay depends on the value of the programmable decay constant τ. Once an output spike is fired, the neuron immediately resets to the voltage level 0.

The architecture was implemented with the XSG toolbox and a discrete time step of 0.125 ms was chosen for these simulations. The maximum clock speed is defined implicitly which depends on the propagation delay of the components used in the design. In XSG, the computational blocks receive inputs and produce outputs at every clock cycle. The Xilinx blocks were assigned computation latencies in order to match paths which have to be simulated in parallel. XSG blocks have default latencies associated with each block. Individual latencies from each computational block were calculated to balance the paths that have to be simulated in parallel. In order to assign the clock period for simulation, worst case delay of the circuit was calculated. The fixed point simulations were used where total numbers of bits were defined along with the binary points. The fixed point format provides flexibility in the number of bits used to represent a number. It is not area efficient to use the same fixed point representation for all the blocks in the design, therefore a format has to be chosen which is good enough to provide required precision and accuracy. For this implementation, both synapses and neuron (membrane) were represented with 18 bits in the Fix_18_12 bit format. Other blocks such as LFSR, comparators, constant values and register delays were represented with different precision formats in order to save area as shown in Table II.

A trade-off has to be made in precision and area where higher precision will cost more area and less precision could cause errors. In order to test the spiking behaviour of implemented architectures as explained by equation 6 and 7, the VHDL code was generated with Xilinx ISE design suite and synthesised for FPGA implementation where different hardware resources and maximum frequency was calculated. The design was targeted for Virtex-II Pro device (xc2vp50) with a speed grade of 5. A single neuron with two synapses took 85 slices out of 23,616. The design could run with a maximum clock speed of 74 MHz after default optimisation process within XSG toolbox. FPGA runs slower than a maximum clock speed because of single cycle implementation. Another reason for slower clock speed achieved on FPGA is due to automated code generation from XSG blocks and in-efficient mapping of global and local routing lines for internal connections and In/Out ports. The synapses were modelled without multipliers; however, one embedded multiplier will be required to model exponential decay of a leaky membrane. A total of 680 slices and 8 multipliers is required for a reservoir of size 8 neurons with 16 synapses. It takes only 8 slices to implement two synapses and if the total number of synapses were increased to 100, it will take 400 slices. If the synapses were modelled with traditional multiplication technique, then a reservoir of size 8 neurons with 16 synapses requires 24 embedded multipliers (16 for synapses and one for leaky membrane for each neuron) and by increasing the number of synapses the requirement for multipliers will increase linearly and the maximum number of synapses will be limited by the maximum number of embedded multipliers.

The proposed design completely avoids the multipliers for synapses and regardless of the number of synapses only one multiplier will be required per neuron. It is possible to optimise the speed of a network by either increasing the frequency of the clock or increasing the step size. The maximum frequency allowed in a design is restricted by the maximum delay in a combinational path which is also termed as the worst case delay. The overall speed can be improved by breaking some of the longest combinatorial paths and introducing some registers. The overall speed can also be improved by increasing the step size, however care must be taken to analyse the details of the design so that spike activity is not missed during the intervals of time steps. The proposed architecture offers an alternative solution for implementing one big cortical column on a single device or several compact fully parallel columns.

IV. PROOF OF CONCEPT

The proposed reconfigurable architecture described in section IV was tested with the TI46 dataset [15]. In order to validate the functionality of the neural reservoir, an integrated HW/SW co-design was used where signal pre and post-processing was performed in software and the reservoir (cortical columns) was implemented on hardware as depicted in Fig. 7. The input speech signals were preprocessed to remove silence parts and features were extracted with the technique of Linear Predictive Coding. These features were further converted into Poisson spike trains to be processed as an input stimulus. The spike trains were used as inputs to the reservoir and different states were recorded for post-processing. One state corresponds to the membrane potentials of all the neurons in the reservoir. In this experiment where a reservoir of size 8 neurons is used, eight membrane potentials were recorded in one state. In total, 20 spike trains were generated for one digit and 200 spike trains were processed through the reservoir for a total of 10 digits. The ‘readout’ neurons (feed forward network) were implemented in software for the classification of input digits.

The network is constructed in a way that each neuron was connected with a minimum of two inputs where input spikes were weighted through fixed weights. In order to interface the input spike trains with the neural reservoir, they have to be converted into Simulink Boolean type through input gateways. These input spike trains were used to perturb the reservoir and the responses were collected in terms of membrane potentials and stored in MATLAB workspace for backend classification. The reservoir has to be simulated for the total time steps equivalent to the time steps of spike trains in order to feed input data into the reservoir. Once the total states were recorded, they were further sampled and only five states were recorded for post-processing, the states were recorded in linear scale from start to the end of states. The readout neurons were trained offline until the algorithm converged to the goal and tested with the test samples to evaluate their classification accuracy.

As shown in Fig. 8 that a three-layered recurrent neural reservoir (3x2x3) was implemented where input vectors were directly connected to the neuron cells and each cell had a minimum of two synapses. A total of 8 neuron cells with 16 synapses were implemented. It is possible to increase the number of neurons and synapses with a chain of adders for synaptic accumulation. All the neurons in the reservoir work in parallel because all inputs and corresponding random weights were accessed simultaneously. A total of 16 fixed weights were stored for 8 neurons in the network where each neuron had minimum two inputs. The input stimulus to the reconfigurable reservoir was the Poisson spike trains which were generated off-chip. The reservoir states (membrane potentials) were also stored off-chip for backend classification.

An MLP classifying engine was implemented as a backend in software. Total data was split into two sets: training and testing. One training sample consisted of 40 data points (5 states of total number of eight neurons at five linear time steps from start to the end) and 10 output neurons were used in the output layer which corresponds to the 10 isolated digits. The network was trained with the training samples where each training sample was compared with the target and the accuracy is calculated. The total number of 30 hidden neurons were used which was found to be the best combination with input layer neurons. The overall accuracy drops if the number of hidden layer neurons were increased or decreased to the maximum number of 30 neurons. After testing with the test data and different hidden layers an overall accuracy of 98% was achieved on test data and 100% accuracy was achieved on training data set. Different standard back propagation training algorithms were tested but best results were achieved when the network was trained with the MATLAB Levenberg-Marquardt training algorithm. The training took 124 seconds and converged to the goal after 25 iterations.

V. Conclusion

This paper has presented a design of HW/SW paradigm for developing a reservoir based approach on reconfigurable platform in order to simulate, analyse and implement the inherent parallelism of cortical neural networks. This optimised reconfigurable hardware architecture was carried out at the network level and resources were calculated. The results demonstrated that area-efficient synapse processing is possible and the multipliers required for synapse implementation can be avoided. It was also found that the proposed architecture is scalable and can easily be scaled on multiple FPGAs to form distributed compact parallel columns. With these attractive features, several advantages such as size reduction of the circuit and elimination of control circuitry can be achieved. Moreover, each cell is implemented as standalone computing unit and interconnected with neighbouring neurons. These advantages make it possible to design and implement large self-contained neural reservoirs for sensory fusion tasks on reconfigurable platforms.

References

[1] Maass, W, Natschläger, T and Markram, H (2002), Real-time computing without stable states: A new framework for neural computation based on perturbations, Neural Computation, 14(11):2531-2560.

[2] Jaeger, H & Haas, H (2004), Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. Science, 304(5667), 78–80.

[3] Verstraeten, D., Schrauwen, B., D'Haene, M., Stroobandt, D (2007), An experimental unification of reservoir computing methods, Neural Networks 20, pp. 391- 403.

[4] Joshi P and Maass, W (2004), Movement generation and control with generic neural microcircuits, BIO-ADIT, 2004.

[5] Yin. J, Meng. Y, and Jin. Y (2012), A developmental approach to structural self-organisation in reservoir computing, IEEE transactions on autonomous mental development, Vol. 4 (4), pp. 273 – 289.

[6] Kristof Vandoorne1 et al (2011), Advances in Photonic Reservoir Computing on an Integrated Platform, International Conference on Transparent Optical Networks, pp. 1-4

[7] Yin J and Meng Y (2012), Reservoir Computing Ensembles for Multi-Object Behavior Recognition, IEEE World Congress on Computational Intelligence, pp. 1-8.

[8] Skowronski, MD and Harris, JG (2007), Automatic speech recognition using a predictive echo state network classifier. Neural Networks 20(3): 414-423.

[9] Uysal, I., Sathyendra, H and Harris, JG (2007), Spike based feature extraction for noise robust speech recognition using phase synchrony coding, ISCAS, pp. 1529 – 1532.

[10] Ghani A, McGinnity, TM, Maguire, LP, Harkin, JG (2008), Neuro-inspired speech recognition with recurrent spiking neurons, LNCS 5163, Springer-Verlag, pp. 513-522.

[11] A Ghani, et al; (2010), Neuro-Inspired Speech Recognition Based on Reservoir Computing, Advances in Speech Recognition, ISBN 978-953-307-097-1, pp. 164, September 2010.

[12] A Ghani et al; (2009), Neuro-Inspired Reconfigurable Architecture for Hardware/Software Co-design, IEEE international conference on System on Chip, SOCC, pp: 287 - 290

[13] Maguire, LP, McGinnity, T. M, Glackin, B, Ghani, A, Belatreche, A, Harkin, J (2006), Challenges for large-scale implementations of spiking neural networks on FPGAs" Elsevier Journal of Neurocomputing, Vol. 71 (1-3), pp. 13-29.

[14] Ghani, A, McGinnity, T.M, Maguire, L.P, Harkin, JG (2006), Area efficient architecture for large scale implementation of biologically plausible spiking neural networks on reconfigurable hardware, FPL, pp. 1-2.

[15] Doddington, G R and Schalk, T B (1981), Speech recognition: Turning theory to practice, IEEE Spectrum, 18 (9), pp: 26-32.

[16] Alexander, O'Shaughnessy, D. (1998), Linear predictive coding. pp. 29–32.

[17] Rabiner, L.R and Schafer, R W (1980), Digital processing of speech signals, The Journal of the Acoustical Society of America, Vol 67 (4), pp. 1406-1407. (Levinson Durbin)

[18] Gupta A, Wang Y, Markram H (2000), Organizing principles for a diversity of GABAergic interneuron and synapses in the neocortex, Science 287: 273–278.

[19] Braitenberg, V and Schuz, A (1991), Anatomy of the Cortex: Statistics and Geometry" NY: Springer-Verlag.

[20] Holmgren C, Harkany T, Svennenfors B, Zilberter Y (2003), Pyramidal cell communication within local networks in layer 2/3 of rat neocortex. J. Physiol. 551: 139–

153.
[21] Foldy C, Dyhrfjeld-Johnsen J, Soltesz I (2005), Structure of cortical microcircuit theory, J. Physiol. 562: 47–54.
[22] Yoshimura Y, Dantzker JLM, Callaway EM (2005), Excitatory cortical neurons form fine scale functional networks. Nature 433: 868–873.
[23] Maass, W and Bishop, C (1999), Pulsed Neural Networks, The MIT Press, Massachusetts.
[24] Gerstner, W., Kistler, W (2002), Spiking Neuron Models - single neurons, populations, plasticity, Cambridge University Press, UK.
[25] Mathworks Inc.: 'MATLAB\Xillinx System User Guide'.
Available: http://www.mathworks.co.uk/fpga-design/simulink-with-xilinx-system generator-for-dsp.html
[25] Mathworks Inc.: 'MATLAB\Xillinx System User Guide'. Available:
[26] Xilinx Virtex II Pro. Hardware, Xilinx Inc.: 'Virtex-II Pro and Virtex-II Pro X FPGA User Guide'.
Available: http://www.xilinx.com/support/documentation/user_guides/ug012.pdf

Authors' affiliations:

All authors are with Engineering, Sports and Science (ESS) academic group, University of Bolton, Bolton, Greater Manchester, BL3 5AB, UK.

List of Table and Figure captions:

Fig. 3: An overview of synaptic interaction amongst different neurons

Fig. 4: Simulation time vs size of the reservoir

Fig. 5: Synapse architecture through pulse counting, fixed weight and AND gate

Fig. 6: Membrane architecture

Fig.7: An overview of hardware/software (HW/SW) environment for reservoir based recognition

Fig.8: Hardware implementation of a neural reservoir (3x2x3)

TABLE I
TEST ACCURACY WITH TRAINING SAMPLES=150 AND TEST SAMPLES =50

Hidden neurons	**Test accuracy (%)**
10	56
20	62
25	72
30	89
35	72

TABLE II
BIT RESOLUTION FOR DIFFERENT HARDWARE BLOCKS

Blocks	**Bit resolution**
Adder	Fix_18_12
Accumulator	Fix_18_12
LFSR	Fix_6_6
3 bits shift right operation	Fix_5_3
Threshold value (constant)	Fix_12_8

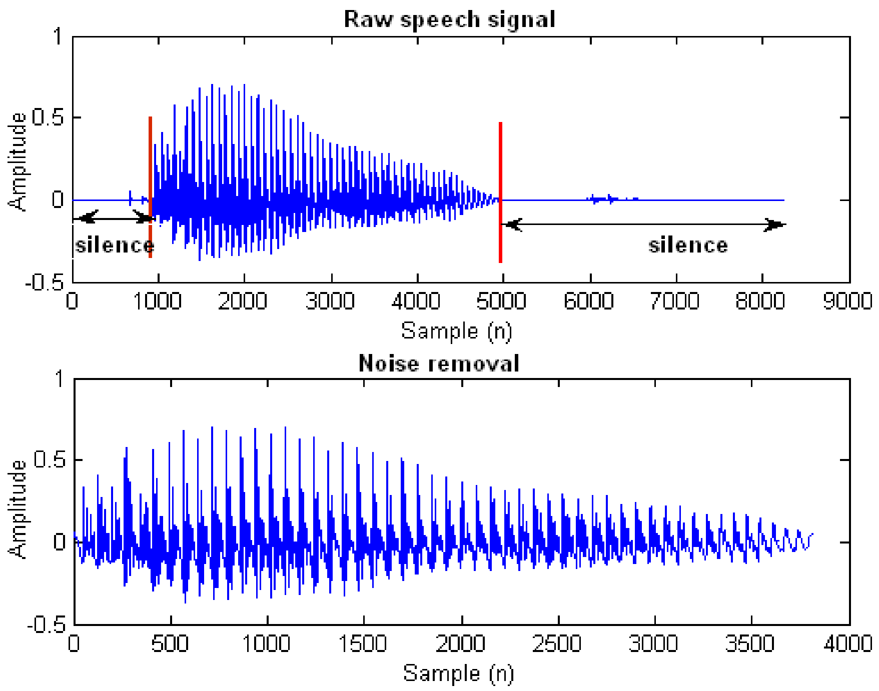

Fig. 1 Raw speech signal (top) and silence removal (bottom)

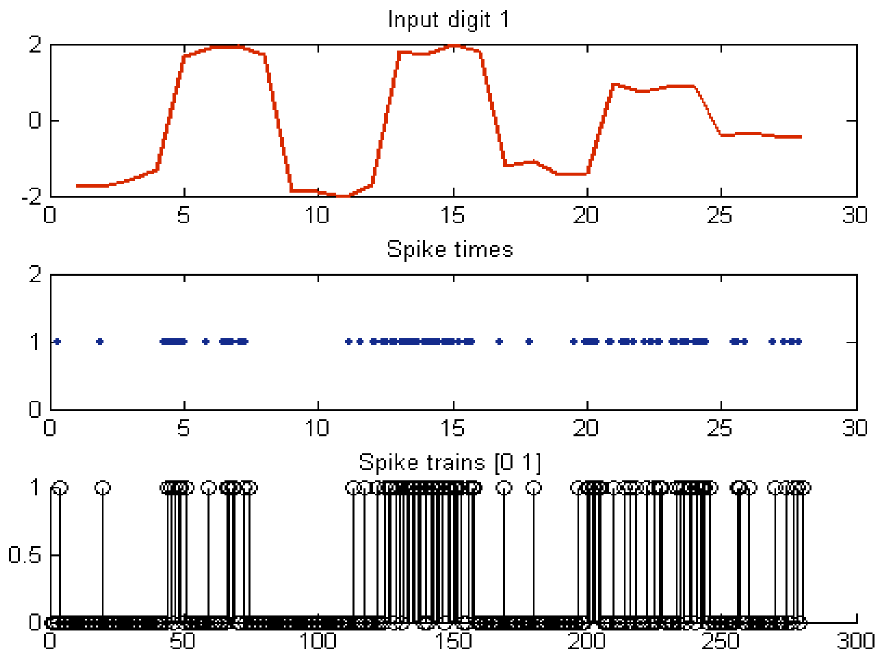

Fig. 2. Input digit ‘1’ and corresponding spike times and spike trains

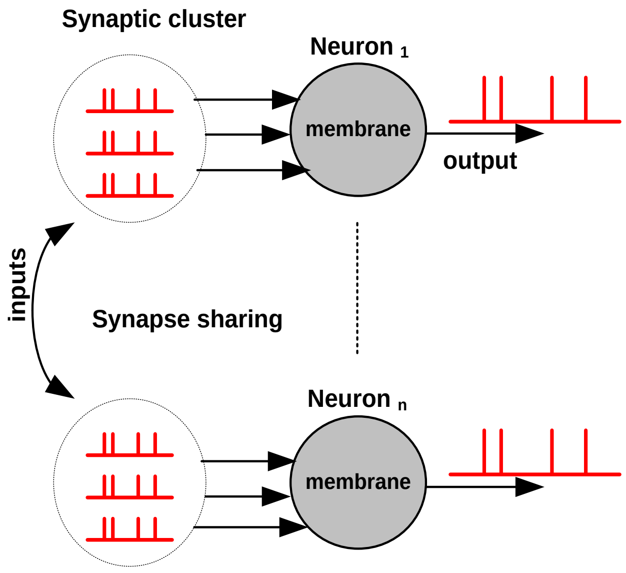

Fig. 3. An overview of synaptic interaction amongst different neurons

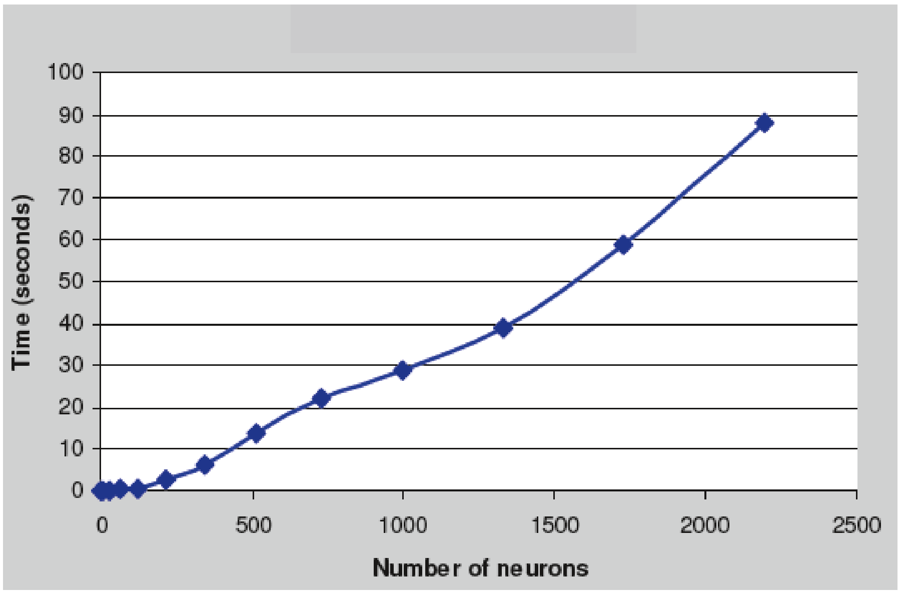

Fig. 4. Simulation time vs size of the reservoir

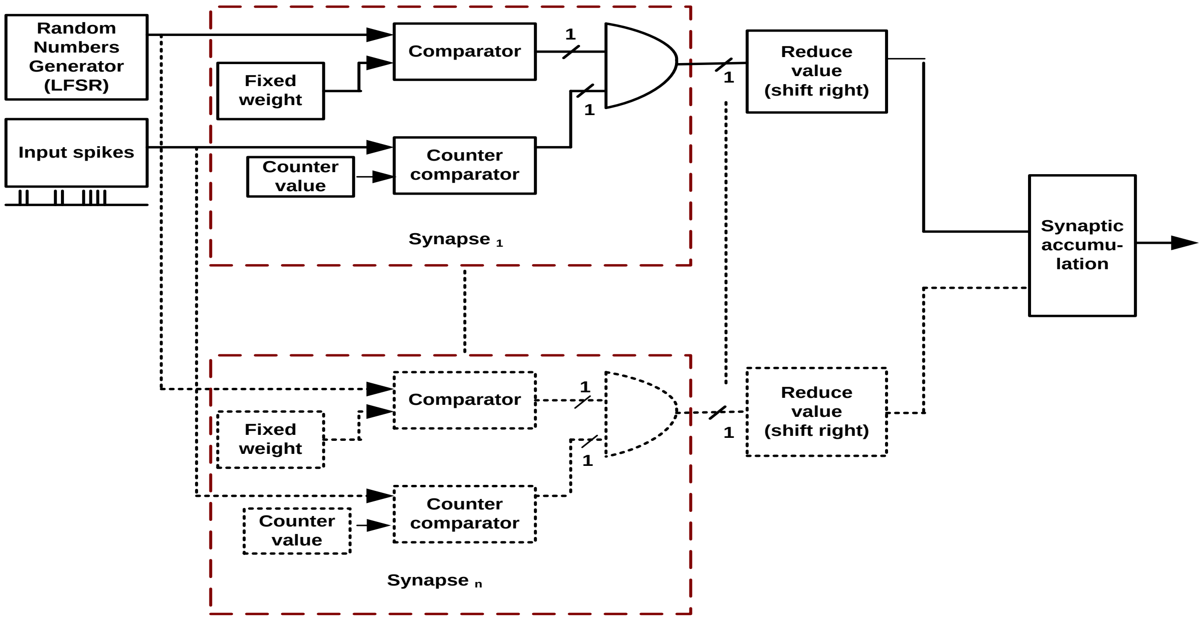

Fig. 5. Synapse architecture through pulse counting, fixed weight and AND gate.

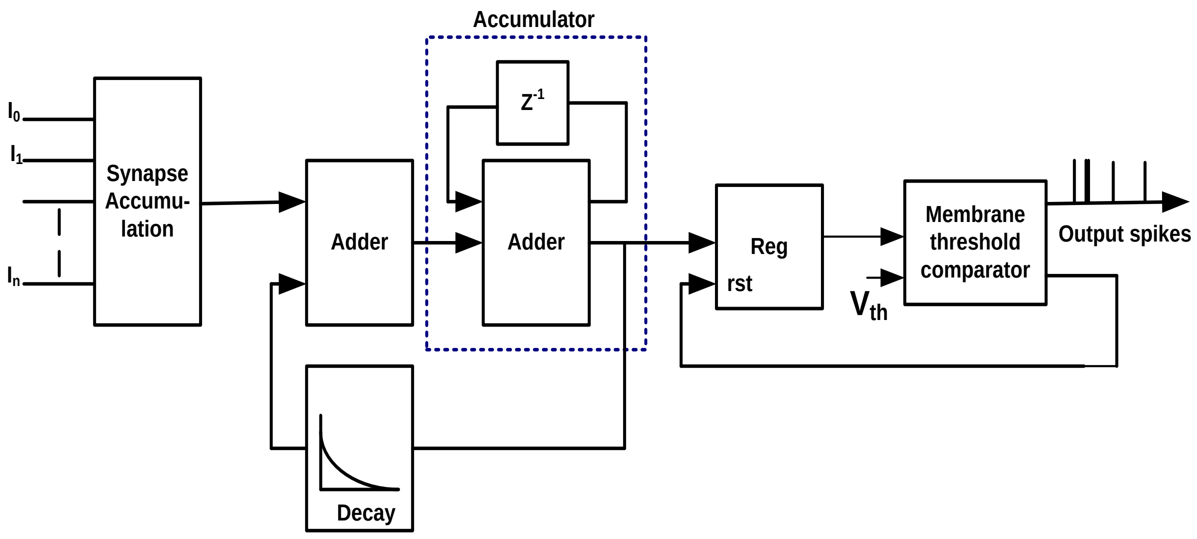

Fig. 6. Membrane architecture

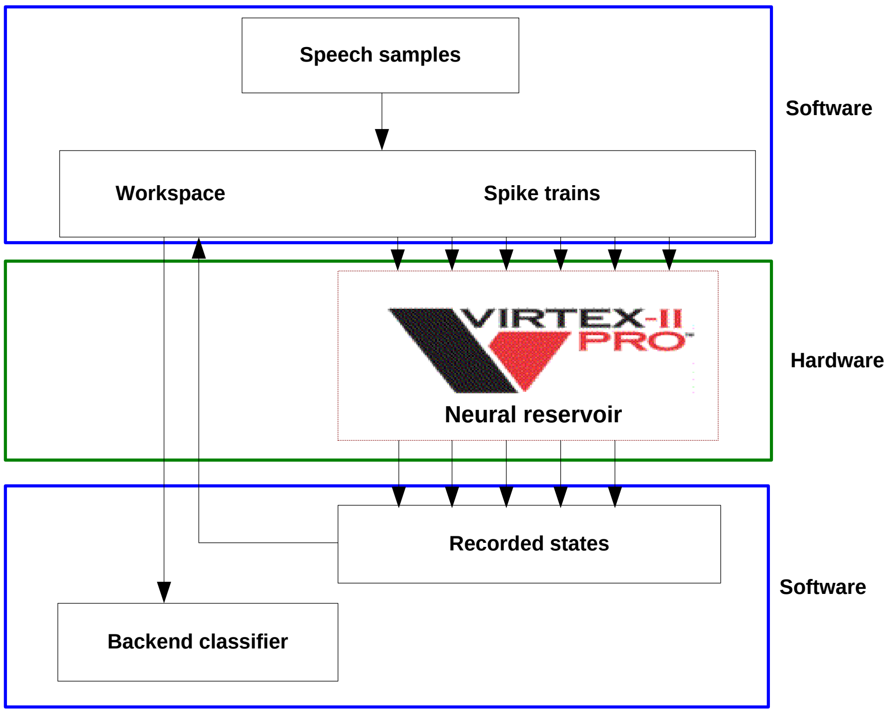

Fig. 7. An overview of hardware/software (HW/SW) environment for reservoir based recognition

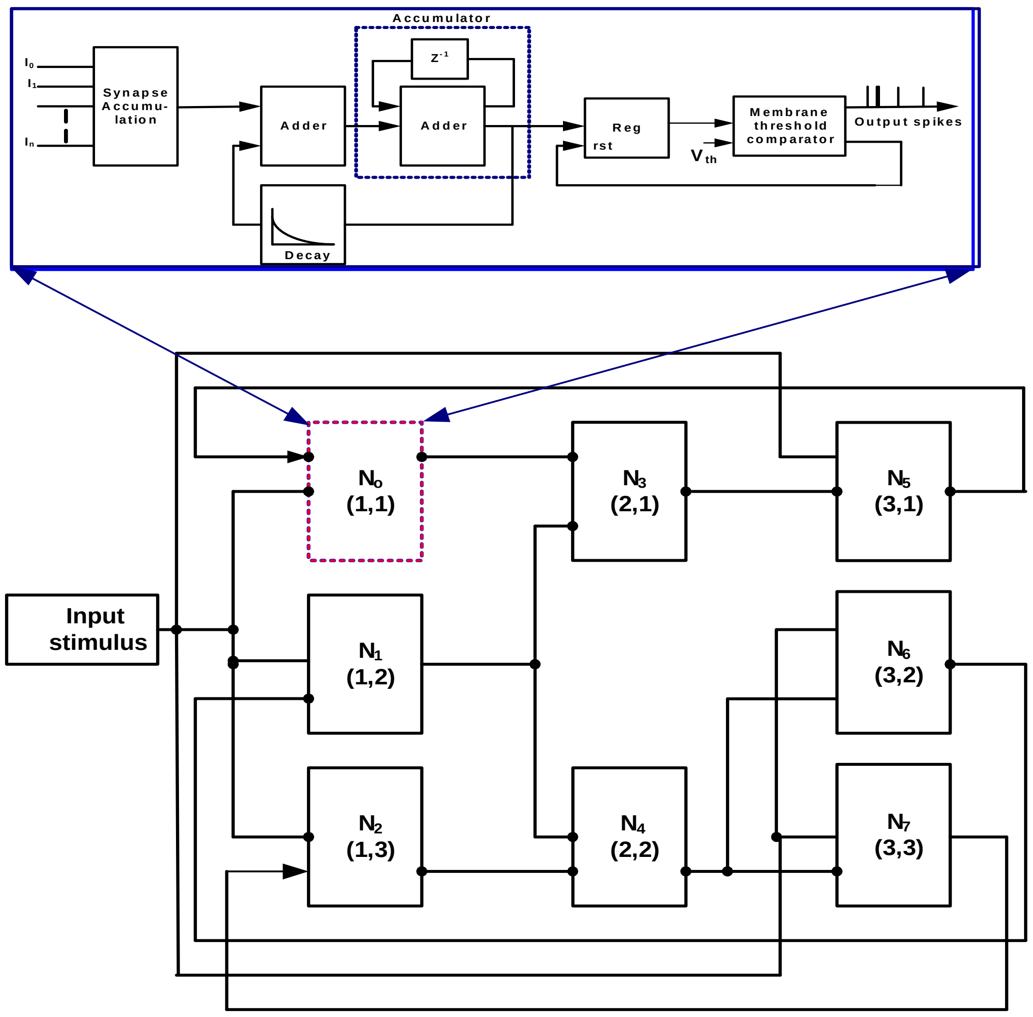

Fig. 8. Hardware implementation of a neural reservoir (3x2x3)