

Compact hardware for real-time speech recognition using a Liquid State Machine

Benjamin Schrauwen, *Student Member, IEEE*, Michiel D’Haene, *Student Member, IEEE*,
David Verstraeten, *Student Member, IEEE*, and Jan Van Campenhout, *Member, IEEE*

Abstract—Hardware implementations of Spiking Neural Networks are numerous because they are well suited for implementation in digital and analog hardware, and outperform classic neural networks. This work presents an application driven digital hardware exploration where we implement real-time, isolated digit speech recognition using a Liquid State Machine (a recurrent neural network of spiking neurons where only the output layer is trained). First we test two existing hardware architectures, but they appear to be too fast and thus area consuming for this application. Then we present a scalable, serialised architecture that allows a very compact implementation of spiking neural networks that is still fast enough for real-time processing. This work shows that there is actually a large hardware design space of Spiking Neural Network hardware that can be explored. Existing architectures only spanned part of it.

I. INTRODUCTION

Many digital hardware architectures for the simulation of Spiking Neural Networks (SNNs) have recently been presented in literature. The main reasons for this is that SNNs, neural network models that use spikes to communicate, have (1) been shown theoretically [1] and practically¹ [2], [3] to computationally outperform analog neural networks, (2) are biologically more plausible, (3) have an intrinsic temporal nature that can be used to solve temporal problems, and (4) are well suited to be implemented on digital and analog hardware. SNNs have been applied with success to several application such as face detection [4], lipreading [2], speech recognition [5], autonomous robot control [6], [7] and several UCI benchmarks [8].

The main drawback of SNNs is that they are difficult to train in a supervised fashion mainly because the hard thresholding that is present in all simple spiking neuron models, making the calculation of gradients very prone to errors which deteriorate the learning rule’s performance [8], [9], [3]. One way to circumvent this is by using fixed parameters. This is what is embodied by the Liquid State Machine (LSM) concept [10] (which is conceptually identical to Echo State Networks [11] and which are generally termed Reservoir Computing [12]). Here a recurrent network of spiking neurons is constructed where all the network parameters (interconnection, weights, delays, ...) are random items. This network, the so called liquid or reservoir, typically exhibits complex non-linear dynamics in its high-dimensional internal

state. This state is excited by the network input, and is expected to capture and expose the relevant information embedded in the latter. As is the case with kernel methods, it is possible to extract this information by processing these network states with simple linear techniques to obtain the actual regression or classification output.

Digital, parallel hardware implementations of SNNs give a large speed-up compared to sequential simulation on a classic processor. Several architectures have already been presented, ranging from large systems [13], [14], [15] able to process very large networks ($> 10,000$ of neurons) at many times real-time speed, to compact designs where small networks are directly mapped onto hardware [6], [16], [17], [18] using small neuron processing elements (PEs). These implementations span a small part of a larger design space which allows to make a trade-off between area (chip area and memory footprint) and calculation time. Given the constraints of the application, which can be speed, chip area, memory usage or power dissipation, we have to choose an architecture that is best suited. Note that in this paper we focus on hard real-time processing thus techniques as event-based simulation are not well suited.

Recently a very convincing engineering application for the Liquid State Machine was presented: isolated spoken digit recognition [5]. When optimally tweaked [19], it can outperform state-of-the-art hidden markov model based recognizers. The system is biologically motivated: a model of the human inner ear is used to pre-process the audio data, next an LSM is constructed with biologically correct settings and interconnection [10], and a simple linear classifier is used to perform the actual classification.

In this paper we present an application oriented design flow for LSM-based hardware implementation. Real-time, single channel speech recognition with the lowest hardware cost is desired. To attain this goal we implement the speech task on two existing hardware architectures for SNNs: a design that processes synapses serially and which uses parallel arithmetic [6], [16] and a design that processes the synapses in parallel, but does the arithmetic serially [17], [18]. As we will show, these architectures are always much faster than real-time, and thus waste chip area. We present a new architecture that uses both serial synapse processing and serial arithmetic. Using this option we are able to process just fast enough for real-time with a very limited amount of hardware. Without much extra hardware cost this design allows to easily scale between a single PE which performs slow serial processing of the neurons to multiple PEs that each process

Benjamin Schrauwen, Michiel D’Haene, David Verstraeten and Jan Van Campenhout are with the Electronics and Information Systems Department, Ghent University, Belgium (email: Benjamin.Schrauwen@UGent.be).

¹The linearly non-separable XOR function can be represented by a single spiking neuron.

part of the network at increased speed. The design space for hardware SNNs has thus been drastically enlarged. The LSM idea (but with threshold logic neurons) was previously already implemented in analog VLSI hardware in [20].

Note that in this paper we will only look at the actual reservoir in hardware, pre-processing the audio and doing the actual classification is still done in software. However, we did implement these blocks in hardware but did not add them to focus this work on the reservoir implementation. All presented designs were implemented at our lab and run on actual hardware.

II. APPLICATION: ISOLATED DIGIT SPEECH RECOGNITION

The isolated digit speech recognition application that will be implemented in hardware is organized as follows: a much used subset of the TI46 isolated digit corpus consisting of 10 digits uttered 10 different times by five female speakers was preprocessed using Lyon’s passive ear model (a model of the human inner ear)[21]. The multiple channel result of this preprocessing step is converted to spikes using BSA [22], a fast spike coding scheme with a good signal to noise ratio. The resulting spike trains are fed into a randomly generated network of spiking neurons, whose parameter settings are optimized using a Matlab toolbox for RC simulations designed at our lab². The responses of the network (the spikes emitted by the neurons) are converted back to the analog domain using an exponential low-pass filter to mimic the operation of a neuron membrane as described in [23] and then resampled using a time-step of 30 ms. The resulting time series are used as input to 10 linear classifiers, one for each digit, which are trained with a one-shot learning approach based on a pseudo-inverse approach with regularization. Training and testing is done in a 10-fold cross-validation setup. The output of the classifiers is post-processed by taking the temporal mean of each classifier’s output and applying winner-take-all to the resulting class-vector. The effect of different weight parameter settings were evaluated, where the performance is measured using the word error rate (WER), which is simply the fraction of misclassified digits. A reservoir was selected that attained a WER of around 5% when simulated with a software implementation of the hardware model (which is part of the toolbox) which takes the quantization effects into account.

The reservoir consists of 200 spiking neurons, all with the same threshold and reset values (255 and -255 respectively) and both absolute and relative refractoriness. The input weights are randomly chosen as either -0.1 or 0.1 and then scaled by the value of the neuron threshold. The internal weights were normally distributed, multiplied by the threshold and rescaled according to the spectral radius (which is a parameter controlling the network dynamics, but which was originally defined for sigmoidal neural networks). The optimal spectral radius was determined to be 0.1. Each

neuron receives 12 input connections, 8 from other neurons in the reservoir and 4 from the input.

III. HARDWARE ORIENTED RC DESIGN FLOW: RC MATLAB TOOLBOX

In the following section, we will present some rough guidelines on how to generally tackle an engineering problem using hardware Reservoir Computing. This was the same flow we used to design the hardware speech application. The RC toolbox offers a user-friendly environment to do a thorough exploration of certain areas of the parameter space, and to investigate some optimal parameter settings in a software environment before making the transition to hardware. The following steps are advisable:

Generic network and node settings. Reservoirs are determined by a broad range of topology and neuron parameters that can influence the performance of the networks significantly. Using the RC toolbox, optimal settings for a given task can flexibly be determined and evaluated. Much of the parameter tuning is as yet empirical and dependent on experience with reservoir computing techniques. Experimental indications for good parameters are described in [12] for many reservoir types, and in [24] for Echo State Networks specifically.

Readout pipeline. Once an optimal reservoir is found, the post-processing of the reservoir states can further improve performance. The RC toolbox offers a number of nonlinear and filter operations that can be applied to the output of the regression step. These simple operations can also greatly influence performance, especially for highly temporal regression tasks.

Evaluate node quantization effects. When the transition is made from software to hardware, computation is changed from floating-point arithmetic to fixed point arithmetic with a tunable precision. This introduces a trade-off between memory and hardware requirements on the one hand, and quantization noise on the other hand. These quantization effects can also be modeled and simulated using the RC toolbox, which allows the user to take these effects into account when designing the reservoir.

Generate network with hardware constraints. Most hardware designs for SNNs take up less hardware area if the neuron structure is as regular as possible (i.e. every neuron has the same number of inputs), because then only a single controller for the entire network is needed instead of one controller per PE. Using the RC toolbox, reservoirs with a regular structure can be constructed a priori, so the user can evaluate the influence of these constraints on the performance. If necessary, an iterated approach of the first four steps is possible.

Export to hardware. Once the optimal reservoir has been determined, the RC toolbox offers helper functions that allow the reservoir structure to be exported automatically to a hardware description.

²This is an open source research toolbox containing a complete scala of reservoir based techniques. It is available at <http://www.elis.ugent.be/rct>.

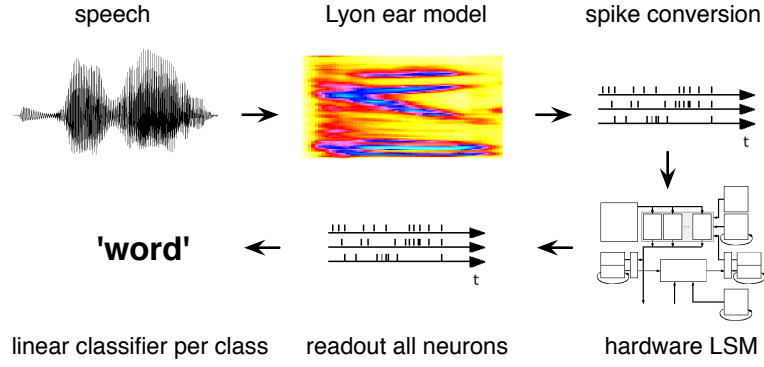


Fig. 1. Overview of the overall speech recognition system.

TABLE I

SPPA HARDWARE RESULTS. THE PARAMETERS FOR THE GENERAL FORMULAS ARE: N NUMBER OF NEURONS, I NUMBER OF INPUTS PER NEURON, B INTERNALLY USED WORD WIDTH, S THE NUMBER OF DISTINCT SYNAPSE TIME CONSTANTS, AND T THE NUMBER OF 'STEPS' USED IN APPROXIMATING EXPONENTIAL DECAY OF THE SYNAPSES AND MEMBRANE (THIS IS USUALLY 1 OF 2).

	4-LUTs	FFs	RAM	Cycles
General	$N(I/2 + 2B + 10)$	$N(1 + B(S + 1))$	$BN(ST + I)$	$(T + 1)(S + 1) + I$
Speech task	13,812	4,038	$56 \times 16,384$	18

TABLE II

PPSA HARDWARE RESULTS

	4-LUTs/SRL16s	FFs	RAM	Cycles
General	$N(22 + 3I + 10S)$	$N(2I + BS + B)$	0	$\lceil \log_2(I) \rceil + B(1 + T) + 1$
Speech task	13,426+2,401	21,743	0	35

IV. EXISTING COMPACT HARDWARE IMPLEMENTATIONS FOR SNNs

The speech recognition application was first implemented on two existing, compact hardware architectures. They will now be briefly introduced and the hardware results for this application will be presented.

A. Serial Processing, Parallel Arithmetic

This architecture [16], [6] processes the neurons as would be done on a classic CPU: by serially processing all synapses and membrane using parallel arithmetic (SPPA). But unlike a classic CPU, each PE processes only one neuron (each neuron is directly mapped on a PE) and the PEs are directly interconnected. We added exponential synapses³ to this architecture, optimized its size and implemented it on a Field Programmable Gate Array (FPGA).

The SNN is simulated using time step based simulation. Each time step consists of several operations, one per clock tick. These operations are: adding weights to the membrane or synapse accumulator, adding the synapse accumulators to the membrane's and decaying these accumulators. Threshold detection and membrane reset is performed in parallel with

these operations. Weights are stored in memory which is located in each PE, while the accumulators are stored in a register bank. When a regular network structure is used (all PEs have the same number of inputs, where some may be unconnected), a single controller can be used to steer the complete network.

When implementing this architecture in FPGA we can derive general (but approximate) scaling formulas for area, memory and time usage with respect to the main network parameters. These are presented in the top row of Table I. For the actual 200 neuron reservoir that is used for the speech recognition application, the implementation results are shown in the lower row (area optimization was turned on for all designs). Note that 56 standard FPGA RAM block of each 16 kbit are used, but they are only partially filled. This design can be clocked at 100 MHz on state-of-the-art FPGAs (Xilinx's XC4VSX35). Given that the incoming speech is sampled at 16 kHz (which is also the rate at which the network operates) we get a network that is 347 times faster than needed for real-time processing. The FPGA is filled for 60 percent.

B. Parallel Processing, Serial Arithmetic

This architecture [17], [18] is based on a parallel synapse processing scheme using serial arithmetic (PPSA) to limit the total size of the PE. Each PE only computes one neuron

³Exponential synapses together with an exponential membrane model result in second order membrane responses. It has been shown theoretically [25] that these models outperform simple first order exponential membrane models.

(direct mapping) and the different PEs are interconnected via direct, fixed connections. Because of the parallel processing scheme, all synaptic weights are stored in a parallel and decentralized way by using many very small memories (we actually use one memory per synapse). This is possible by using FPGA specific features (we use the Xilinx specific SRL16 memories which are 16 bit shift registers that can be implemented in a single 4-LUT).

The parallel processing of the dendritic tree is performed by a binary, direct mapped adder tree where each adder is a one bit serial adder. To improve processing speed we use pipelining⁴.

During each SNN time step, all inputs to the dendritic trees are presented in parallel. It takes $\lceil \log_2(inputs) \rceil$ cycles before the first bit is available at the end of the adder tree. Serially adding this result to the membrane accumulator is performed at one bit per cycle. After this, the membrane is decayed which again is implemented serially and thus takes one bit per clock cycle.

The hardware results for this architecture are presented in Table II. The top row shows the general scaling properties, while the bottom row gives the result specific for the 200 neuron network used for the speech recognition application. This design can be clocked at 115 MHz on the same FPGA and it is filled for 60 percent.. Each simulation time step takes 35 cycles which results in an implementation that is 205 times faster than real-time (with 16 kHz input). Although this design does parallel processing of the dendritic tree, it is slower than SPPA due to the limited number of input. With limited inputs the cost for doing serial arithmetic outweighs the gain made by the parallel dendritic tree. When more inputs are used (approximately 50) PPSA becomes faster than SPPA.

V. MULTIPLE PEs, SERIAL PROCESSING, SERIAL ARITHMETIC

Because both of the previously published architectures give much faster than real-time performance on the speech recognition task, they use more hardware than needed. We will now present a novel architecture for the processing of SNN that allow slower but scalable operations at a highly reduced hardware cost. The architecture processes all synapses serially as well as doing all arithmetic serially (SPSA). This results in a very small implementation of the PE (only 4 4-LUTs!) but in longer computing times. All neuron information is stored in RAM and interconnection between several neurons is also memory based. Due to this, each PE can process several neurons serially, but this at the cost of speed. Because the controller is much larger than the actual PE, we will opt for using one controller and several PEs. To do this, all PEs have to perform the same instructions on different data. This is called a Single Instruction/Multiple Data (SIMD) architecture (the SNN FPGA implementation

presented in [26] is also a SIMD processor but it uses parallel arithmetic). Processing the network consists of two phases: updating the neuron states and performing the interconnection. We chose to do both operations in parallel (unlike [26] which uses two separate phases).

The general setup is that we use one controller, several PEs and each PE processes several neurons per simulated time-step. We will call the time needed to simulate a single neuron a cycle. Each simulated time-step thus consists of several cycles. A simplified control flow is shown in Figure 2. The global cycle structure of weight adding, decay and threshold testing is quite similar to the other architectures, but now each of these command is processed serially, i.e. that they consist of several clock cycles.

An overview of the system is given in Figure 3a. Several processing elements connect parallelly to 4 memories where each memory gets an address from the controller. The synapse and membrane memory is the working memory of the PEs holding the synapse and membrane accumulators (and some constants). Note that for each PE this is actually a one bit memory, but because all PEs get the same instructions the addresses to these memories are the same and can thus be implemented as one multi-bit memory. The second memory is a circular buffer holding the weights. Note that the weight is placed in serially with several weights in parallel for the different PEs. The third and fourth memories are used for input and output spikes to and from the PEs. These are actually double buffered so that the interconnection PE can copy data while the neuron PEs are running. The interconnection PE is organized in such a way that receiver-oriented copying is performed: at cycle T , the interconnection PE is copying the input spikes for the neurons processed in cycle $T + 1$ into the second buffer of the input spike memory. At the next cycle, these two memories are switched. Spike output memory is switched after each complete simulated time-step. External input and output is possible via a memory interface. Note that all spike copying is done on a bit by bit basis. Both the input and output spike memories are two-port memories which have a one-bit and a multi-bit port.

The actual neuron PE data-pad can be seen in Figure 3b. Although it might seem quite complex, it can fit in 4 4-LUTs. The PE uses a 3-port memory⁵ which hold the membrane and synapse model potential, and all constants such as decay factors, threshold and absolute refractory. The data path of the PE is centered around a one-bit adder. One port is directly connected to the adder while the other adder input can be a weight or one of the ports of the 3-port memory, which can then be negated, forced to one or forced to zero. The output of the adder is fed to a bypass multiplexer to allow the inputs to bypass the adder. The result is written back to the 3-port memory. Some of the control signals are directly generated by the controller while others are mediated through a flip-flop

⁴After each tree operation, memory elements are added. This allows higher clock speed, but requires multiple clock cycles before the result is available at the output.

⁵Three-port memory is not a primitive component of FPGAs but can easily be emulated thanks to the high speed memories (memories are 5 times faster than our actual design). This allows the memory to run at double speed and convert a two-port memory into a four-port memory by time-multiplexing.

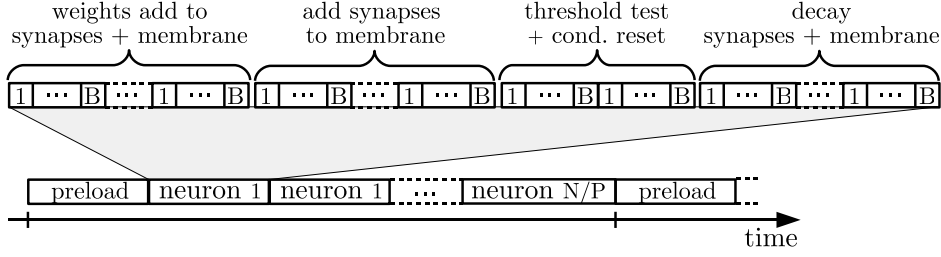


Fig. 2. SPSA timing

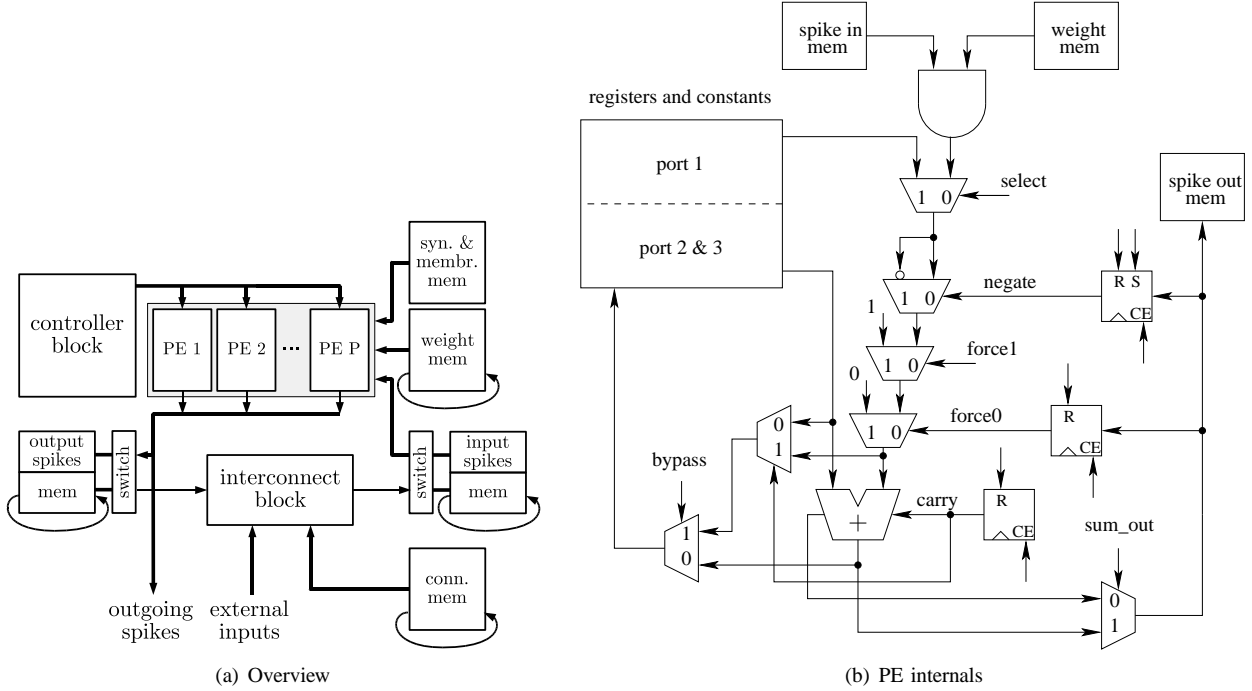


Fig. 3. SPSA overview and PE internals

which has a clock enable and a synchronous set and reset. The actual control signals and their timing behavior will not be elaborated on for brevity.

An overview of the approximate size results for the neuron PE, the controller and the interconnection is given in Table III, where P stands for the number of PEs. The number of clock ticks per cycle is equal to $(S + 3 + B(I + 2S + 5))(\lceil N/P \rceil + 1)$. The first part is the number of clock ticks needed per cycle, while the second part is the number of cycles. Notice that one cycle is added to allow pre-loading by the interconnection block of the input spikes for the first cycle.

The results for the speech application are summarized in Table IV. Note that some of the memory is implemented in LUTs using LUT-RAM, the column denoting 'clock cycles' represents the number of clock cycle per simulated time-step, and 'real-time' denotes how much faster than real-time processing that option is. We tested an increasing number of PEs which results in a large final speed variation, however with very little space variation. Optimal settings for real-time processing are achieved with 5 PEs.

With the current hardware we can maximally and in real-time process approximately 1600 neurons at 16 kHz and this using 40 PEs. When processing larger networks at this speed, the interconnect block becomes a bottleneck.

Note that adding more classes to the classification problem (for example if going from isolated digits to phonemes), the architecture stays exactly the same, only more readout functions need to be calculated. The reservoir implementation and the number of nodes that are read-out stay exactly the same.

VI. CONCLUSIONS

In this work we showed that real-time speech recognition is possible on limited FPGA hardware using an LSM. To attain this we first explored existing hardware architectures (which we reimplemented and improved) for compact implementation of SNNs. These designs are however more than 200 times faster than real-time which is not desired because lots of hardware resources are spend on speed that is not needed. We present a novel hardware architecture based on serial processing of dendritic trees using serial

TABLE III
SPSA GENERAL SIZE RESULTS

	Number	LUTs	FFs	RAM
PE	P	4	3	$B(2S + 6) + 2IP + 2N$
Controller	1	270	$33 + 2 \log_2(\lceil N/P \rceil BI)$	0
Interconnect	1	170	$\log_2(NSP \lceil N/P \rceil^2)$	$IN \log_2(N \lceil N/P \rceil)$

TABLE IV
SPSA SPEECH APPLICATION HARDWARE RESULTS

PEs	4-LUTs LUT-RAMs	FFs	RAM	Clock cycles	Mhz	Times real-time
1	488+40	223	9×16,384	38994	145	0.23
5	489+40	223	9×16,384	7954	145	1.1
10	489+40	223	9×16,384	4074	143	2.2

arithmetic. It easily and compactly allows a scalable number of PEs to process larger networks in parallel. Using a hardware oriented RC design flow we were able to easily port the existing speech recognition application to the actual quantized hardware architecture.

For future work we plan to add the pre-processing (ear model) and the actual classification to the hardware. Both sub-blocks are already implemented at our lab.

ACKNOWLEDGEMENT

This work was partially funded by FWO Flanders project G.0317.05.

REFERENCES

- [1] W. Maass, "Noisy spiking neurons with temporal coding have more computational power than sigmoidal neurons," in *Proceedings of NIPS*, pp. 211–217, 1997.
- [2] O. Booi, "Temporal pattern classification using spiking neural networks," Master's thesis, University of Amsterdam, 2004.
- [3] B. Schrauwen and J. Van Campenhout, "Backpropagation for population-temporal coded spiking neural networks," in *Proceedings of IJCNN*, 2006.
- [4] A. Delorme and S. Thorpe, "Face identification using one spike per neuron: resistance to image degradations," *Neural Networks*, pp. 795–804, 2001.
- [5] D. Verstraeten, B. Schrauwen, D. Stroobandt, and J. Van Campenhout, "Isolated word recognition with the liquid state machine: a case study," *Information Processing Letters*, vol. 95, no. 6, pp. 521–528, 2005.
- [6] D. Roggen, S. Hofmann, Y. Thoma, and D. Floreano, "Hardware spiking neural network with run-time reconfigurable connectivity in an autonomous robot," in *NASA/DoD Conference on Evolvable Hardware*, pp. 189–198, 2003.
- [7] D. Floreano, J. Zufferey, and J. Nicoud, "From wheels to wings with evolutionary spiking neurons," *Artificial Life*, 2004.
- [8] S. M. Bohte, H. L. Poutre, and J. N. Kok, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, pp. 17–37, November 2002.
- [9] B. Schrauwen and J. Van Campenhout, "Extending spikeprop," in *Proceeding of IJCNN*, 2004.
- [10] T. Natschlager, W. Maass, and H. Markram, "Real time computing without stable states: A new framework for neural computation based on perturbations," *Neural Computation*, vol. 14, no. 11, pp. 2531–2560, 2002.
- [11] H. Jaeger and H. Haas, "Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless telecommunication," *Science*, vol. 308, pp. 78–80, April 2 2004.
- [12] D. Verstraeten, B. Schrauwen, M. D'Haene, and D. Stroobandt, "A unifying comparison of reservoir computing methods," *Neural Networks*, 2006. Submitted.
- [13] A. Jahnke, U. Roth, and H. Klar, "A SIMD/dataflow architecture for a neurocomputer for spike-processing neural networks (NESPINN)," in *Proceedings of MicroNeuro*, pp. 232–237, 1996.
- [14] T. Schoenauer, N. Mehrtash, A. Jahnke, and H. Klar, "MASPINN: Novel concepts for a neuro-accelerator for spiking neural networks," in *VIDYNN'98 – Workshop on Virtual Intelligence and Dynamic Neural Networks*, (Stockholm), 1998.
- [15] H. Hellmich, M. Geike, P. Griep, P. Mahr, M. Rafanelli, and H. Klar, "Emulation engine for spiking neurons and adaptive synaptic weights," in *Proceedings of IJCNN*, (Montreal, Canada), pp. 3261–3266, 2005.
- [16] A. Upegui, C. A. Peña Reyes, and E. Sanchez, "An FPGA platform for on-line topology exploration of spiking neural networks," *Microprocessors and Microsystems*, vol. 29, pp. 211–223, 2005.
- [17] B. Schrauwen and J. Van Campenhout, "Parallel hardware implementation of a broad class of spiking neurons using serial arithmetic," in *Proceedings of ESANN*, pp. 623–628, 2006.
- [18] B. Girau and C. Torres-Huitzil, "FPGA implementation of an integrate-and-fire LEGION model for image segmentation," in *Proceeding of ESANN*, pp. 173–178, 2006.
- [19] D. Verstraeten, B. Schrauwen, and D. Stroobandt, "Reservoir-based techniques for speech recognition," in *Proceeding of IJCNN*, 2006.
- [20] F. Schürmann, K. Meier, and J. Schemmel, "Edge of chaos computation in mixed-mode vlsi - "a hard liquid", in *Proceeding of NIPS*, 2004.
- [21] R. Lyon, "A computational model of filtering, detection and compression in the cochlea," in *Proceedings of the IEEE ICASSP*, pp. 1282–1285, May 1982.
- [22] B. Schrauwen and J. Van Campenhout, "BSA, a fast and accurate spike train encoding scheme," in *Proceedings of IJCNN*, pp. 2825–2830, 2003.
- [23] R. Legenstein and W. Maass, *What makes a dynamical system computationally powerful?* MIT Press, 2005.
- [24] H. Jaeger, "Tutorial on training recurrent neural networks, covering bptt, rtrl, ekf and the "echo state network" approach," Tech. Rep. GMD Report 159, German National Research Center for Information Technology, 2002.
- [25] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, 1997.
- [26] M. J. Pearson, C. Melhuish, A. G. Pipe, M. Nibouche, I. Gihlesphy, K. Gurney, and B. Mitchinson, "Design and FPGA implementation of an embedded real-time biologically plausible spiking neural network processor," in *Proceeding of FPL*, pp. 582–585, 2005.