DESIGN AND FPGA IMPLEMENTATION OF AN EMBEDDED REAL-TIME BIOLOGICALLY PLAUSIBLE SPIKING NEURAL NETWORK PROCESSOR

*M.J.Pearson *, C.Melhuish, A.G.Pipe, M.Nibouche, I.Gilhesphy, K.Gurney, B.Mitchinson*

IAS laboratory
University of the West of England
Coldharbour Lane, Bristol,
BS16 1QY
email: martin.pearson@uwe.ac.uk

ABSTRACT

The implementation of a large scale, leaky-integrate-and-fire neural network processor using the Xilinx Virtex-II family of Field Programable Gate Array (FPGA) is presented. The processor has been designed to model biologically plausible networks of spiking neurons in real-time to assist with the control of a mobile robot. The real-time constraint has led to a re-evaluation of some of the established architectural and algorithmic features of previous spiking neural network based hardware. The design was coded and simulated using Handel-C Hardware Description Language (HDL) and the DK3 design suite from Celoxica. The processor has been physically implemented and tested on a RC200 development board, also from Celoxica.

1. INTRODUCTION AND BACKGROUND

Spiking Neural Networks (SNNs), or Pulse Coded Neural Networks (PCNNs), differ from many other neural networks in that inter-neural communication is reduced to temporally separated spikes or pulses. This coding scheme is analogous to biologically observed neural codes [1]. Models of real neural systems using SNNs have been proposed by neuroscientists in the past using computer based software simulations [2]. Typically, these models will not be tested in real-world situations and therefore remain in the theoretical domain. To test them in a real world situation, as in mobile robotics for example [3], [4], requires a processing modality which can maintain real-time performance and be embedded on a platform with limited physical space and power resources. FPGAs fulfil this specification and have the added advantage of design tools which facilitate the rapid translation of software algorithms directly into hardware. This allows them to take advantage of increased processing throughput, derived from parallelisation techniques and hardware optimisation, at a fraction of the cost of full Application Specific Integrated Circuit (ASIC) implementation.

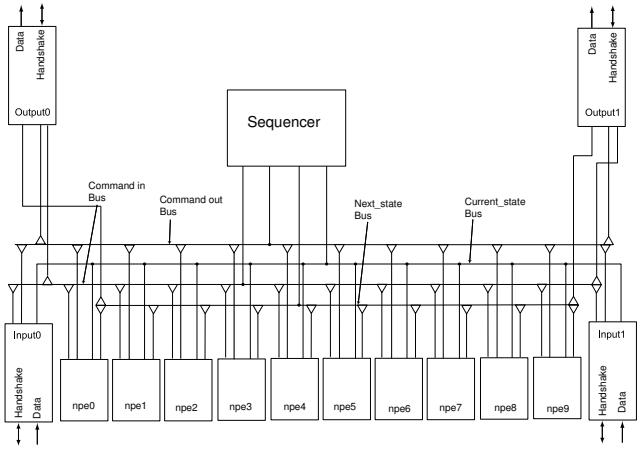

Fig. 1. Block diagram of processor topology

The design presented in this paper is best described as a 'soft-core' Single Instruction path, Multiple Data path (SIMD) array processor. Hardware based SNN processors have been developed [5],[6], however, these approaches are best described as 'hardware accelerators', or co-processors, since they cannot guarantee hard-real-time performance at all levels of neural activity. This is due to an event driven approach to network state update which takes advantage of the sparse inter-neural connectivity and low average activity levels of biological neural networks. When modelling large networks with high peak network activity, they will not be able to continuously update the network in real-time. This situation is unacceptable in control systems, particularly distributed control systems, which highlights the principle reason that the processor detailed in this paper was designed.

*Collaboration with Adaptive Behaviour Research Group, University of Sheffield and funded by EPSRC as part of the *whiskerbot project* Grant No. GR/S19639/01

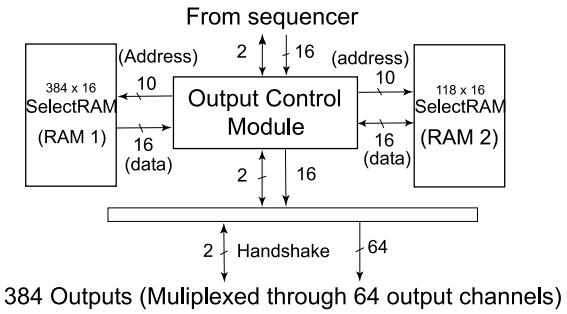

Fig. 2. Block diagram an output module

2. THE PROCESSOR

The processor consists of an array of Neural Processing Elements (NPEs) and two pairs of input/output modules operating concurrently from the same instruction set as illustrated in Fig.1. A central sequencer issues the instructions, which actually constitute synchronisation cues due to the processing algorithm of each module being explicitly implemented in the hardware. The input/output modules can interface asynchronously across different external clock domains using a simple handshaking protocol. The SIMD neural processor has 10 NPEs, each of which emulate 112 'virtual' neurons and 912 synapses using time division multiplexing. The update period of the processor is set at 500μS, which constitutes a real-time update period for biologically plausible SNNs [7].

The hardware for the processor was described and captured using the C syntax Hardware Description Language (HDL) Handel-C. The Celoxica DK3 development environment was used to compile and synthesize Handel-C to EDIF for target specific physical synthesis by Xilinx Project Navigator. The development board used is the Celoxica RC200, which has a XC2V1000 Xilinx Virtex-II FPGA at its core and various peripheral devices for rapid project prototyping.

2.1. Module specifics: Sequencer

The sequencer maintains real-time performance and coordinates the concurrently operating modules of the processor. An operational iteration (epoch) of the processor consists of two phases; the update phase and the communication phase. The sequencer initiates the update phase at the beginning of an operational epoch. It then monitors all the modules and initiates the communication phase only when the update phase has completed. The communication phase involves each active module being polled to release the contents of its current state memory (i.e. the results of the preceding update phase) and instructing other modules to store this in their next state memory space (i.e. the data to be used in the next update phase). A real-time counter is used to regulate the initiation of each operational epoch.

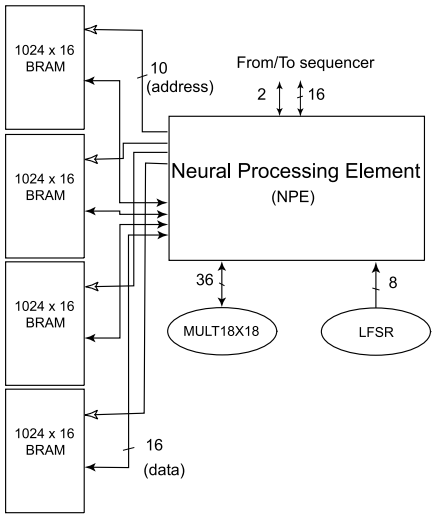

Fig. 3. Block diagram of neural processing element

2.2. Module specifics: Input modules

The input module has 64 input channels which can be connected to physical pins or an internal interface using the configurable logic array of the host FPGA. The handshaking lines facilitate asynchronous operation and allow communications across different clock domains. During the update phase of an epoch, the state of the 64 input channels are transferred onto the internal 16 bit data bus of the module and stored in the current state memory. The 64 channels are each read 6 times during the update phase, giving a total of 384 inputs per module. During the communications phase all 384 of the stored input states are broadcast to the rest of the processor.

2.3. Module specifics: Output modules

The output module, shown in Fig.2, is similar in both architecture and operation to the input module. The communication protocol is the same; utilising 2 handshaking lines and a multiplexed 64 channel output port generating 384 outputs. The source address of each of the outputs are stored in a block of distributed, or SelectRAM (RAM 1 see Fig.2). The next state memory (RAM 2) contains the current state of the entire network, i.e., the activity of all inputs and neurons. This is updated during the communication phase of an operational epoch and it is from here that each of the outputs are read, as indexed by the source addresses in the user defined memory block.

2.4. Module specifics: Neural Processing Element

This module contains a hardware implementation of a neuron and a single synapse. The contextual information of 112 virtual neurons and 912 synapses are stored locally in

4 banks of BlockRAM, as illustrated in Fig.3. The context for each neuron and synapse are sequentially multiplexed onto the hardware during the update phase of each epoch. A copy of the state of the entire network is stored locally in each NPE (as in the output module) which serves as the input stimulus for the virtual neurons/synapses. The updated state of each of the neurons in the NPE are stored in the local next state memory space and are broadcast to the rest of the processor during the communication phase. The hardware optimised 18×18-bit signed multipliers (MULT18X18) are used to implement fixed-point integer arithmetic operations such as exponential decays and multiplicative noise. Noise can be simulated on both the synaptic weight and the membrane threshold using a simple 16-bit, Linear Feed forward Shift Register (LFSR) acting as a pseudo-random number sequence generator within each NPE. The random numbers generated are used to index elements within a memory space containing two pre-defined noise distributions. During trials a Gaussian distribution was used for generating threshold noise and a Rayleigh distribution was used for synaptic noise. The reason these particular noise distributions were chosen was to match empirical electro-physiological data taken from real neurons.
Fig.4 details a block diagram of the neuron and synapse model used in the NPE. The inclusion of noise and inter-neural propagation delays to these models emphasises the biological plausibility of the system.

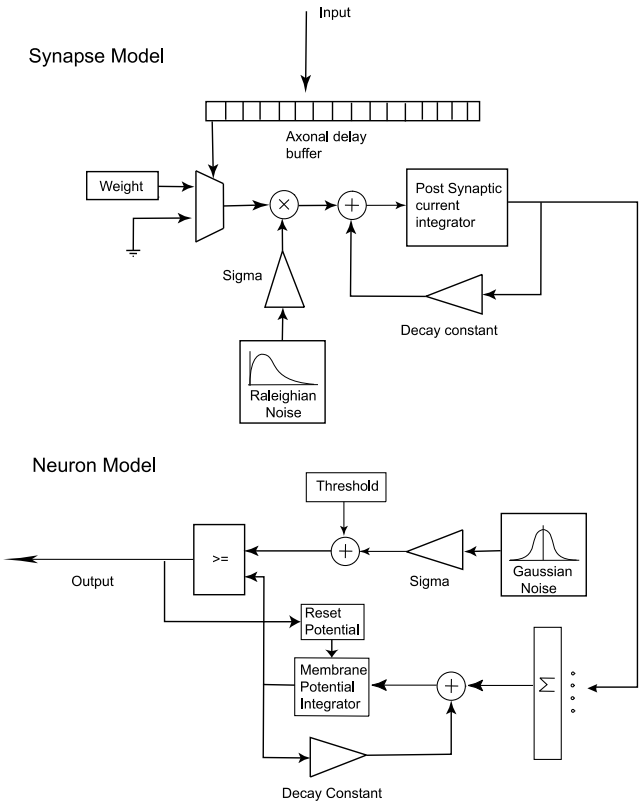

Fig. 4. Block diagram of individual neuron and a single synapse

2.5. Discussion

The disadvantages of using a local network state memory in each module of a parallel processing system are well documented [7], [1]. These disadvantages are particularly apparent when implementing SNNs with biologically plausible activity rates. However, the principle disadvantage of using a more efficient, central network state memory is the memory access latency introduced by bus arbitration. This can lead to unacceptable performance in a real-time application and has therefore not been adopted in this processor. Each synapse and output is associated with a pre-synaptic source address, stored as a parameter in a local contextual memory space. If a target orientated addressing scheme were used (as in [5]) the time period of the communication phase will be dependent on the topology and activity of the network. During periods of high activity the time required for the communication phase may cause the duration of the operational epoch to exceed real-time. Using the source orientated addressing scheme with a local network activity memory space, guarantees a fixed time period for the communication phase independent of network activity. This is a high priority in the specification for this processor and is therefore the approach which has been adopted.

3. DEVELOPMENT CYCLE

The project in which this processor has been developed is a collaboration between neuroscientists and engineers. There is a requirement, therefore, for an interface to be established between the two parties through which models and results can be passed freely. A set of universal file formats have been adopted which contain network parameters and interconnect information, input test stimuli and the corresponding output of the defined network. A conventional SNN software simulator is used by the Neuroscientists to prototype potential neural structures and generate network parameter lists. These parameters are translated into a format which can be used to initialise the memories of the FPGA based neural processor. This translation includes converting from a floating point number system into a 16-bit fixed point representation. A C++ coded simulator of the FPGA based SIMD processor is used to rapidly assess the degradation in model integrity as a result of this translation. Similar tests can also be conducted using the Handel-C simulator though at a greatly reduced simulation speed. A simulation of 2 seconds of real-time FPGA based processor activity requires approximately 5 hours of simulation time using Handel-C. To establish that the behaviour of the processor at a physical

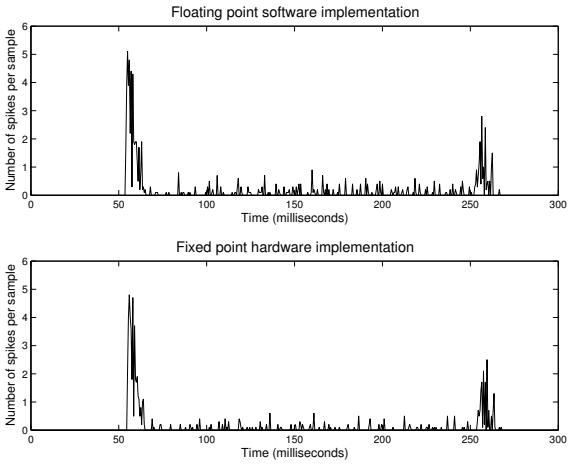

Fig. 5. Comparative Peri-Stimulus Time Histogram (PSTH) plots of 10 presentations of the same stimulus to an 1100 neuron SNN implemented using a floating point software processor and fixed point hardware processor. (Correlation coefficient = 0.7662)

level is the same as indicated in simulation, a test platform has been implemented on an FPGA using a neural processor and a serial port interface. These two components are linked together so that the processor core can be loaded with the universal input stimulus file. The resulting output is then externally logged and converted into the standard output file format for comparison. Finally the bit file that was generated to configure the test FPGA neural processor core, can be used to configure an embedded FPGA and form part of a control system.

4. RESULTS

As part of the development cycle described above, existing models of real neural structures were used to test and synchronise the two software FPGA simulators (implemented in C++ and Handel-C). Upon synchronisation the two fixed point simulators generated indistinguishably similar output and internal state matches in response to all corresponding sets of input stimulus.

Most of the observed discrepancies between the floating point and fixed point models were due to differences between the random number generators in the two systems. The statistics of the random number distributions from both generators are, however, very similar. This was highlighted by repetitive presentations of the same input stimulus to both hardware and software processors using randomly seeded generators as shown in Fig. 5. Other discrepancies were found to be introduced by quantisation distortion of certain parameters when translated from floating to fixed point. This problem was overcome by pre-scaling such parameters and introducing a small amount of noise (or 'dither') to arithmetic operations to remove the bias. The inclusion of these two techniques has resulted in the observed behaviour of the simulated hardware processor matching almost exactly that of the floating point software simulator.

Finally, the processor required a series of fully embedded tests to confirm that its operation, as indicated by the Handel-C simulator, can be replicated in real hardware. All the results generated from these tests confirm that the processor, implemented on an FPGA, performs in real-time exactly as has been indicated by the simulation.

5. CONCLUSION

It has been demonstrated that a large biologically plausible SNN model can be implemented in hard real-time using a single FPGA. Use of Handel-C HDL to describe this architecture facilitated both the rapid development time and the subsequent synchronisation procedure to generate an intermediate software based hardware simulator. This simulator eases the transfer of models developed using software SNN simulator packages, using a floating point number system, into a fixed point integer representation implemented in hardware for use in real-time, real-world control applications.

6. REFERENCES

[1] W. Maass, "Computing with spiking neurons," in *Pulsed Neural Networks*, W. Maass and C. M. Bishop, Eds. MIT Press (Cambridge), 1999, pp. 55–85.

[2] K. Gurney, T. Prescott, and P. Redgrave, "A computational model of action selection in the basal ganglia i: A new functional anatomy," *Biological Cybernetics*, vol. 84, pp. 401–410, 2001.

[3] R. R.I.Damper and T.W.Scutt, "Arbib: An autonomous robot based on inspirations from biology," *Robotics and Autonomous systems*, vol. 31, pp. 247–274, 2000.

[4] The, "Whiskerbot project," *EPSRC grant number: GR/S19639/01*. [Online]. Available: http://www.whiskerbot.org

[5] J. Schemmel, K. Meier, and E. Mueller, "A new vlsi model of neural microcircuits including spike time dependent plasticity," in *Proceedings of IJCNN'04*. IEEE Press, 2004, pp. 1711–1716.

[6] N. Mehrtash, D. Jung, H. Hellmitch, T. Schoenauer, V. Lu, and H. Klar, "Synaptic plasticity in spiking neural networks (sp^2inn): A system approach," *IEEE Transactions on Neural Networks*, vol. 14, no. 5, 2003.

[7] C. Koch and I. Segev, Eds., *Methods in Neuronal Modeling: From Synapses to Networks*. Cambridge, Massachusetts: MIT Press, 1989.