A Generic Building Block for Hopfield Neural Networks with On-Chip Learning

Michael Gschwind, Valentina Salapura, Oliver Maischberger
{*mike,vanja,oliver*}*@vlsivie.tuwien.ac.at*

Institut für Technische Informatik
Technische Universität Wien
Treitlstraße 3-182-2
A-1040 Wien
AUSTRIA

Abstract

We present an extendable digital architecture for the implementation of a Hofield neural network using field-programmable gate arrays (FPGAs). Due to its bit-serialk implementation, the actual digital circuitry is simple and highly regular, thus allowing efficient space usage of FPGAs. We exploit the reprogrammability of these devices to support on-chip learning.

1 Introduction

This paper discusses a hardware implementation of a Hopfield neural network with on-chip learning using FPGAs. In the past, neural networks were often simulated on general purpose computing machines, which led to less-than-satisfying performance. Several ASIC implementations have been proposed [LM93] [LL93], but using FPGAs, specialized nets can be designed when only a few implementations are needed.

In our past research, we have concentrated mainly on implementations based on off-chip learning. While on-chip learning is a nice feature, we have felt that it is not really mission critical when much of a net's life time is spent processing. However, for short life-time applications, learning time may dominate the entire life span. On-chip learning is also a very useful when a fast debug cycle is required. The circuit is designed such that the on-chip learning hardware constitutes only a small portion of the entire logic.

2 The Hopfield Neural Network

The Hopfield neural network is a model of associative content addressable memory with a simple flexible structure [KH90]. As a content addressable memory, it is capable not only of storing information, but also to carry out certain computational tasks such as error correction and nearest neighbor search.

The Hopfield neural network consists of N completely connected neurons arranged in one layer. The i-th neuron can be in one of two states: $u_i = -1$ (off) and $u_i = 1$ (on). The actual state of a Hopfield neural network with N neurons is described by the state vector $\vec{u}$, the N dimensional binary (± 1) vector, whose i-th component u_i corresponds to the state of the i-th neuron. The strength of the synaptic connection from neuron j to neuron i is given by a real value w_{ij}.

The total input v_i to neuron i is:

$$v_i = \sum_{\substack{j=1 \\ j \neq i}}^{N} w_{ij} u_j \tag{1}$$

Depending on the value of total input and a threshold value t_i, the neuron changes the value of its output or leaves it fixed according to an activation function of neurons. This activation function for a two-state neuron is a hard limiter. For a Hopfield neural network, the threshold value is typically $t_i = 0$.

The strength of a synaptic connection is determined by the Hebbian rule. For K N-dimensional binary (± 1) column vectors $\vec{u}_i$ which are to be stored in a Hopfield neural network it is defined as matrix $\vec{W}$:

$$\mathbf{W} = \sum_{i=1}^{K} \left(\vec{u}^i \vec{u}^{iT} - \mathbf{E} \right) \tag{2}$$

where $\mathbf{E}$ denotes the $N \times N$ identity zero matrix and superscript T denotes transposition to a row vector. $\mathbf{W}$ is $N \times N$ a zero-diagonal symmetric matrix whose en-

tries w_{ij} determine the strength of the synaptic connection from neuron j to neuron i. The synaptic strength matrix $\mathbf{W}$ and threshold vector $\vec{t}$ fully define the neural network.

3 Implementation

In our implementation, the threshold for all neurons is set to 0, so the network is fully defined by its weight matrix which is stored in RAM. The width of this RAM is a function of the network size, as the capacity of a Hopfield net is $N/(2 * \mathrm{ld}N)$. With a net size of 64 neurons, this gives a capacity of 5.3 vectors which could be stored during the training phase. Since training for each input vector can change a single weight matrix element w_{ij} by at most ± 1, the value range for w_{ij} is $[-5.3, 5.3]$. This would require a 4 bit wide RAM.

To reduce interconnect requirements, a neuron cell has only 2 input and 2 output ports. One input and output port are available to set and read the state of a particular neuron from an external controller. The other two ports form a ring shift register and are connected to the preceeding and succeeding neural cell, respectively. This shift register is used to pass the results from the previous iteration to all neurons, so as to compute u_i for the next cycle. The neuron also contains an additional D flip-flop to store its current state.

To implement the weight matrix RAM, we use the RAM cells available in the Xilinx XC4000 series. The D-flip flops for the shift register and the current state are implemented using CLB flip flops [Xil93].

There are four distinct operational modes for a Hopfield neural network. These phases are:

- initialization,
- training,
- classification, and
- read-out.

3.1 Initialization

During the initialization phase, all synapse weights are reset to 0. A global counter addresses all RAM locations and resets them. This operation is necessary whenever a new training phase is to be performed.

3.2 Training

The training phase implements the Hebbian learning rule (equation 2), which can be rewritten as follows:

$$\mathbf{W} \leftarrow \mathbf{W} + \left(\vec{u}\vec{u}^T - \mathbf{E}\right) \qquad \text{for all input vectors } \vec{u}. \tag{3}$$

To train the neural network, a new pattern to be recognized is loaded into the neurons. These data are loaded into both the shift register and each neuron's state register.

To perform the vector multiplication, the training pattern is shifted through all N neurons of a net. With each shift, each neuron i reads out the corresponding w_{ij} from the weight RAM. It than multiplies its current state with the values presented to it through the shift register, adds the result to w_{ij} and writes back w_{ij} to the weight RAM. After N shift operations, the new pattern has been stored into the weight matrix $\mathbf{W}$.

The multiplication of two values is inherently simple, as the values are restricted to ± 1, with -1 represented by a low signal and $+1$ by a high signal. Thus, this operation can be represented by a 1 bit comparator.

The part of the hardware implementation responsible for the learning phase of the neuron is shown in figure 2. The RAM locations are addressed by a counter which is incremented on each shift. This counter is implemented as global unit, and not duplicated in each neuron block.

3.3 Classification

In the classification phase, the pattern to be recognized is loaded into the shift-register of the neural network. Each neuron multiplies all N inputs u_j by the corresponding synaptic weights w_{ij}, accumulates the results of this multiplication, and applies the activation function.

Multiplication and accumulation are implemented by an adder/subtracter block. Since u_j can only be -1 or 1, multiplication is achieved by steering the function of the add/subtract unit with the value u_j. After N cycles, the new state of the neuron is derived by applying the SGN activation function.

4 Design Results

We have realised this design in a 64 neuron configuration; each neuron requires 26 CLBs on an XC4000 series Xilinx FPGA, for a grand total of 1664 CLBs for the whole neural net. Thus, this design can be implemented with 3 XC4013 parts.

Timing is primarily dependent on the speed of the adder/subtracter block, and is on the order of 25 MHz. With 64 shifts required for a complete iteration, we achieve a firing rate of 390000 per second. The overall network performance is 25M connections per second, irrespective of the number of neurons used in a network.

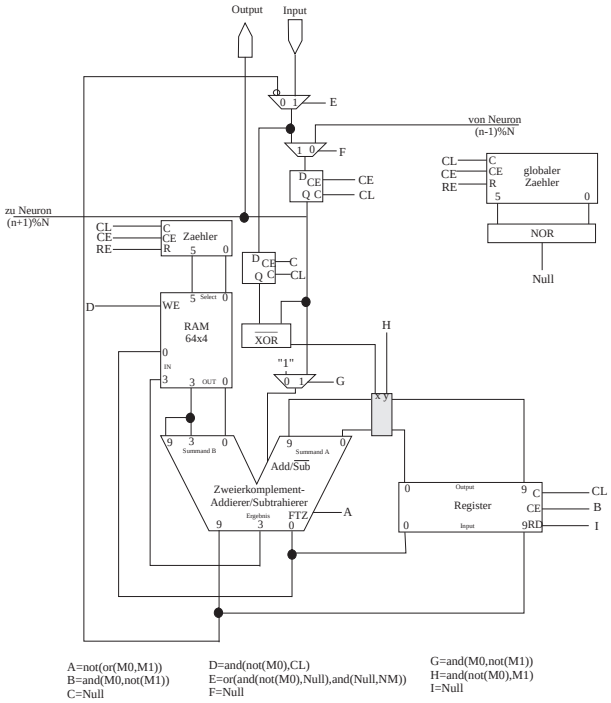

Figure 1: Schematic of a neuron cell.

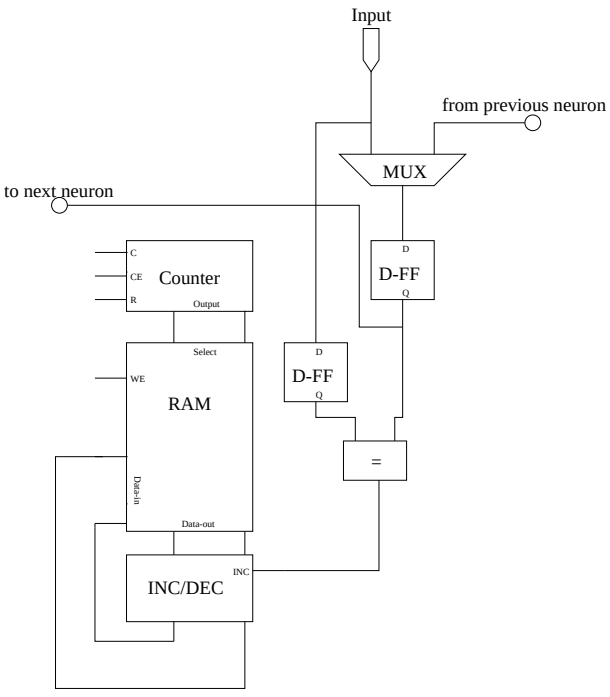

Figure 2: Learning phase of the Hopfield neural network.

5 Related Work

GANGLION [CB92] is an implementation of a simple three layer feed forward net. GANGLION uses a highly parallel, pipelined architecture, with the aim of providing high performance. This design uses two Xilinx XC3090 FPGAs and a 2Kx8 PROM per neuron, trading high area consumption for high performance. Ferrucci [Fer94] shows how the GANGLION approach can be extended to self-adapting networks.

Marchesi et al. [MOPU93] reduce hardware complexity by restricting weights to numbers of the form $2^n + 2^m$ – thus two shift registers and an adder suffice to perform the multiplication by the synaptic weight. However, this design cannot be used with conventional training algorithms.

In [SGM94], we propose a neuron design using a digital representation which can be implemented with a fraction of the logic required for GANGLION, by using only one multiplier which is shared by all synaptic weight computations.

van Daalen et al. [vDJST93] use a stochastic approach to neural net implementation. They represent values v in the range $[-1, 1]$ by stochastic bit streams in which the probability that a bit is set is $(v + 1)/2$. Their input representation and architecture restrict this approach to fully interconnected feed-forward nets. The non-linear behavior of this approach requires that new training methods be developed.

6 Conclusion

We have presented a space efficient implementation of a binary Hopfield neural network with on-chip learning capabilities. The implemented learning algorithm is the Hebbian learning rule, which is used in several different models of neural networks.

This implementation requires only 26 CLBs per neuron, allowing large nets to be constructed. Hopfield neural networks implemented with this design achieve a performance of 25M connections per second.

References

[CB92] Charles E. Cox and W. Ekkehard Blanz. GANGLION – a fast field-programmable gate array implementation of a connectionist classifier. *IEEE Journal of Solid-State Circuits*, 27(3):288–299, March 1992.

[Fer94] Aaron Ferrucci. A field-programmable gate array implementation of a self-adapting and scalable connectionist network. Master's thesis, University of California, Santa Cruz, January 1994.

[KH90] Yves Kamp and Martin Hasler. *Recursive Neural Networks for Associative Memory*. John Wiley and Sons, Chichester, UK, 1990.

[LL93] John A. Lansner and Thorsten Lehmann. An analog cmos chip set for neural networks with arbitrary topologies. *IEEE Transactions on Neural Networks*, 4(3):441–444, May 1993.

[LM93] Harold J. Levy and T. C. McGill. A feedforward artificial neural network based on quantum effect vector-matrix multipliers. *IEEE Transactions on Neural Networks*, 4(3):427–433, May 1993.

[MOPU93] Michele Marchesi, Gianni Orlando, Francesco Piazza, and Aurelio Uncini. Fast neural networks without multipliers. *IEEE Transactions on Neural Networks*, 4(1):53–62, January 1993.

[SGM94] Valentina Salapura, Michael Gschwind, and Oliver Maischberger. A fast FPGA implementation of a general purpose neuron. In *Proc. of the Fourth International Workshop on Field Programmable Logic and Applications*, Prag, Czech Republic, September 1994.

[vDJST93] Max van Daalen, Peter Jeavons, and John Shawe-Taylor. A stochastic neural architecture that exploits dynamically reconfigurable FPGAs. In *IEEE Workshop on FPGAs for Custom Computing Machines*, Napa, CA, April 1993. IEEE CS Press.

[Xil93] Xilinx. *The Programmable Logic Data Book*. Xilinx, Inc., San Jose, CA, 1993.