Space Efficient Neural Net Implementation

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 show how field-programable gate arrays can be used to efficiently implement neural nets. By implementing the training phase in software and the actual application in hardware, conflicting demands can be met: training benefits from a fast edit-debug cycle, and once the design has stabilized, a hardware implementation results in higher performance.

While neural nets have been implemented in hardware in the past, larger digital nets have not been possible due to the real-estate requirements of single neurons. We present a bit-serial encoding scheme and computation model, which allows space-efficient computation of the sum of weighted inputs, thereby facilitating the implementation of complex neural networks.

1 Introduction

Conventional computer hardware is not optimized for simulating neural networks. Therefore, several hardware implementations for neural nets have been suggested ([MS88], [Sal90], [CB92], [vDJST93]).

While the functions of neural networks are comparatively simple, their hardware implementation is expensive. A neuron can be modeled as

$$n_i(x_1, \ldots, x_{m_i}) = a_i(\sum_{1 \le j \le m_i} w_{ji} * x_j),$$

where x_j are the input signals, w_{ji} the weights and a the activation function.

Neural performance stems from the simultaneous execution of many neurons and their interconnections, not the performance of any single neuron. Thus, the main focus of our project was space efficiency to allow for large nets, not performance.

We decided to use field-programable gate arrays (FPGAs) to develop a prototype of our net ([Xil92]). FPGAs can be reprogrammed easily, thus allowing different design choices to be evaluated in a short time. This design methodology also enabled us to keep overall system cost at a minimum.

Early on we decided on off-chip training, with the fully trained configuration being downloaded into hardware. This reduced real estate consumption for two reasons:

- No hardware is necessary to conduct the training phase.
- Instead of general purpose operational units, specialized instances can be generated. These require less hardware, as they do not have to handle all cases. This applies especially to the multiplication unit, which is a major resource hog in neural net implementation.

As training occurs only once in the lifetime of a neural net application, namely at the beginning of its lifetime, this off-chip training scheme does not present a limitation to a net's functionality. Our choice of FPGAs as implementation technology proved beneficial in this respect, as each net can be trained on the workstation and then down-loaded to the FPGA for operational use.[1]

The neurons as modeled in our prototype can be used for any network architecture. They operate as *binary threshold units* with synaptic weights in the range $[-1, 1]$. The neuron fires if the sum of weighted inputs exceeds a predetermined threshold. A neuron only takes two states: active (1) or inactive (0).

2 Related Work

Murray and Smith [MS88] demonstrate an analog implementation of a neural net. They use pulse stream encoding for representing values. This facilitates an efficient implementation of multiplication by intersecting the input signal with a high-frequency chopping signal.

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.

GANGLION [CB92] is a very fast implementation of a simple three layer feed forward net, but needs 640 to 784 CLBs per neuron. – The neurons described here need only 22 CLBs. For maintaining the high performance of GANGLION a special high-speed bus architecture is necessary, which is rising costs.

3 Multiplication Using Digital Chopping

In neural net implementation, the most complex, involved and space-consuming feature is the weighing of inputs by multiply units. As every neuron has several inputs, several of these operations have to be performed concurrently for a neuron to be fully functional. Thus, we concentrated our optimization efforts on this unit.

By performing the training phase off-chip, considerable space can be saved over an off-chip training scheme, as general-purpose multiplication units can be replaced by units optimized towards multiplying by a specific constant. This improvement did, unfortunately, not prove sufficiently space-conserving to allow the efficient implementation of non-trivial nets.

[1] The FPGA technology here matches a definite need. A silicon implementation of our neural net architecture would have to model part of the FPGA functionality, namely parametrization through RAM cells (if at a more optimized level) to make this approach usable in a general scheme. It is not a viable solution to implement fully trained neural networks in silicon. This would require a new production and validation cycle for every newly trained neural network.

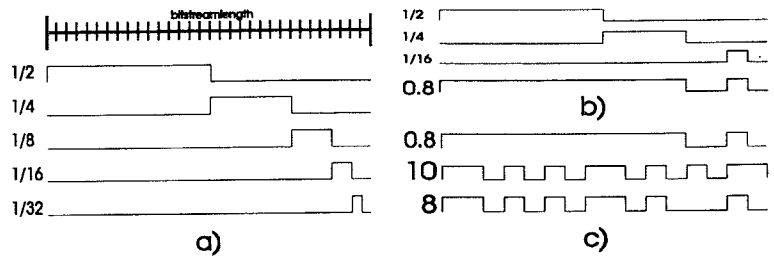

Figure 1: Chopping signals and multiplication: (a) chopping signals for a bit stream of length 32. (b) generation of chopping weights by the union of the basic chopping signals. (c) chopping multiplication by intersection of two signals.

In our efforts for reducing the real estate cost of the multiplication, we came to the conclusion, that changing the encoding of values allows extremely efficient implementation of the weighing of inputs by using *digital chopping.*

Using the time division multiply technique [Ern60], two analog values can be multiplied by intersecting the two signals, provided that one signal has a considerably higher frequency than the other. This process is called chopping and is restricted to multiplication by values from the range $[0, 1]$.

For digital operation, we decided to represent both the input values and the weights as bit streams (see figure 1). Multiplication is performed by intersecting the two streams bit by bit. Thus a weight of 0.5 is represented by a chopping bit stream which has half of its bits set.

In digital operation, the frequency disparity necessary for correct operation is not easy to attain. Since we use hard limiter activation functions, the state of any neuron is restricted to active(1) or inactive(0). These values are encoded by bit streams consisting entirely of either 1s or 0s. This guarantees a sufficient frequency disparity for correct operation.

For the input neurons, values v in the range $[0, 1]$ are encoded by a bit stream where the probability that a bit is set is v. The set bits have to be uniformly distributed over the entire bit stream to ensure correct operation. This requirement is a result of the limited frequency spectrum available in digital circuits. Thus, correctness of operation has to be ensured by probabilistic methods.

4 The Basic Neuron

A neuron weighs its input values in the synapses, accumulates the post synaptic signals, and applies the activation function. As digital chopping is a simple operation, all inputs can be operated on in parallel by separate synapse instances.

The accumulation of post synaptic signals is performed by time multiplexing to reduce hardware complexity. After one processing run, the threshold activation function determines the output value of the neuron. Figure 2 shows a complete neuron with eight inputs: each bit is processed by a synapse and fed to a multiplexer. This multiplexer selects each post synaptic signal and feeds it to a counter which accumulates the post synaptic signals. A sign function determines whether a specific input is inhibiting or activating. This determines the signal on the counter up/down signal, thereby circumventing the restriction of chopping multiplication

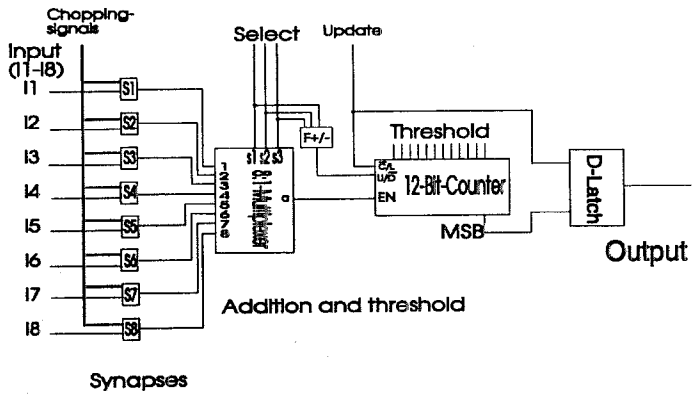

Figure 2: Schematic diagram of a neuron

to the range $[0, 1]$. The counter contains the accumulated value of all post synaptic values in a biased representation: the value 0 is represented by the bit pattern 100000000000_2.

The counter is also used to implement the threshold activation function of the neuron. At the beginning of each computation cycle (i.e. when a new bit stream is fed to the neuron), the counter is initialized by loading the negated threshold value Θ in based representation, i.e. $2048 - \Theta$. Thus, to implement a neuron with threshold 128 the counter will be initialized to $2048 - 128 = 1920$. This value will then be hardwired and not consume any resources. The activation state of a neuron can then be concluded from the counter's most significant bit.[2]

As we use bit streams of length 256, and accumulation of the post synaptic signals requires eight cycles per bit (one cycle for each input channel), a new computation cycle is started every 2048 cycles. This condition is checked by a global counter, and distributed to all neurons. Upon receiving this signal, the neurons will latch their output state and reload the counter to restart a new computation [Mai93].

5 Network Architecture and Performance

The design of the neurons is such that any neural architecture can be assembled from single neurons. Users can choose an optimal interconnection pattern for their particular application, as these interconnections are performed using FPGA routing. Thus, our design can be used to implement, *inter alia*, Hopfield [Hop82], feed-forward, recursive, and Kohonen self-organizing feature maps [Koh90]. Figure 3 shows a feed-forward network with four neurons in the input layer, four neurons in the hidden layer and two neurons in the output layer. A global counter is used to synchronize the neurons.

The design presented in this paper can be scaled to meet varying performance and precision requirements and to support an application-specific number of input signals (see table 1).

Several neurons can be put on one FPGA. The exact number depends on the exact FPGA type, the number of inputs and the required precision, and can be anywhere from eight to 128. By replicating these basic building blocks, arbitrarily large, complex neural nets can be designed

[2] As $2048 - \Theta + \Sigma \geq 2048 \Longrightarrow \Sigma \geq \Theta$!

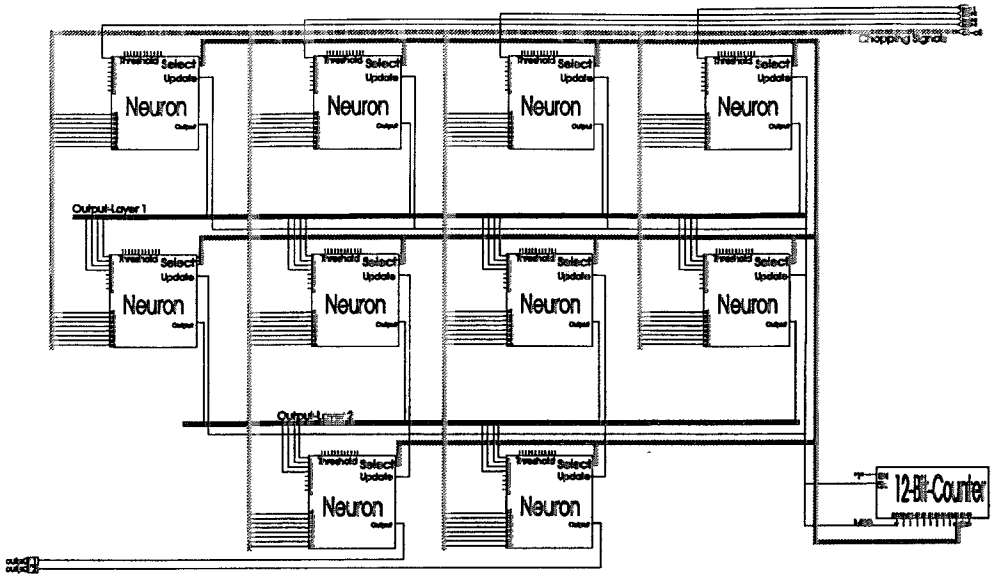

Figure 3: Example architecture: a feed-forward network with ten neurons.

precision	8 inputs	4 inputs
0.0039	16000+	32000+
0.0078	32000+	64000+
0.0156	64000+	128000+

Table 1: Firing rate of one neuron (per second, at 33 MHz), as a function of the number of inputs and value representation precision. Precision is defined as the delta between two consecutive representable values.

cheaply and efficiently. Having neurons as indivisible functional units allows absolute freedom in choosing any topology required.

6 Conclusion

We propose a space-efficient, scalable neural network design, which can be adapted to support a wide range of performance, precision and input requirements. Starting from an optimized, freely interconnectable neuron, various neural network models can be realized.

By using bit stream encoding of values and digital chopping, the hardware required for implementing a single neuron can be reduced considerably. This allows for the massive replication of neurons to build complex neural nets. FPGAs are used as hardware platform, facilitating the implementation of arbitrary network architectures and the use of an off-chip training scheme.

Future projects include the development of a wider variety of neurons (with different activation functions, performance/real estate trade-offs etc.) and the complete automization of

the the design flow from the network architecture definition phase and training to the finished hardware implementation.

7 Acknowledgement

We wish to thank Alexander Jaud for his help with the ViewLogic and the Xilinx design environments.

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.

[Ern60] Dietrich Ernst. *Elektronische Analogrechner – Wirkungsweise und Anwendung.* R. Oldenbourg, München, Deutschland, 1960.

[Hop82] John J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the Academy of Sciences USA*, volume 79, pages 2554–2558, April 1982.

[Koh90] Teuvo Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, September 1990.

[Mai93] Oliver Maischberger. Die Implementation Neuraler Netze mittels FPGAs. Technical report, Technische Universität Wien, Wien, Österreich, 1993. (to be published).

[MS88] Alan F. Murray and Anthony V. W. Smith. Asynchronous VLSI neural networks using pulse-stream arithmetic. *IEEE Journal of Solid-State Circuits*, 23(3):688–697, March 1988.

[Sal90] V. Salapura. A digital Hopfield neural network. In *ETAN Conference*, Zadar, Croatia, June 1990.

[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.* IEEE CS Press, April 1993.

[Xil92] Xilinx. *The XC4000 Databook.* Xilinx Corp., 1992.