VLSI IMPLEMENTATION OF NEURAL NETWORKS WITH APPLICATION TO SIGNAL PROCESSING

JABRI M. PICKARD S. LEONG P. RIGBY G. JIANG J. FLOWER B. HENDERSON P.

University of Sydney, School of Electrical Engineering, Systems Engineering and Design Automation Laboratory

ABSTRACT

Analog sub-threshold MOS techniques offer solutions to Application Specific Integrated Circuit where power consumption is a strict constraint affecting architectural and circuit design decisions. Mapping a functional neural network model to analog sub-threshold is a challenging task, and requires careful architectural, system level and circuit level consideration, with respect to the constraints inherent in this technology. This paper presents our experience in this mapping process. A multi-layer learning algorithm suitable for analog sub-threshold implementation is presented. We also discuss system level issues and describe circuits of neurons and synapses that have been designed, and present fabrication results.

1. Introduction

Analog sub-threshold MOS technology offers solutions to the design and implementation of artificial neural networks where power consumption is a strict constraint. Although many researchers have reported *circuits* operating in sub-threshold mode, very little has been reported on the implementation of *systems.*

The artificial neural network (ANN) systems addressed in this paper are programmable ones, in contrast with sensory-like circuits described by Carver Mead [1], so facilitating learning either on or off chip. This paper considers multi-layer feedforward networks (MLFN), although the techniques can be easily adapted to recurrent networks.

2. Mapping Learning into Silicon

Although back-propagation is an efficient algorithm for training MLFN, it is not a "silicon-wise" learning algorithm, such as Madaline Rule III [2] (node perturbation), because:

- The need for a backward pass through the hardware means the hardware has to be bidirectional.
- The requirement of hardware to compute the back-propagated errors (multiplication by derivatives, *etc*)
- The need to compute a derivative function.

If learning is performed off-chip then the output of each neuron has to be made available, which can mean a serious pinout problem.

The advantages of MRIII over back-propagation are:

- No backward pass is required, so hardware need not to be bidirectional.
- Derivative of neuron output with respect to input is not needed.

So, the backward pass of back-propagation is replaced by an evaluation of the gradient of the error with respect to to the weights using "node perturbation", that is, the relationship $\frac{\delta E}{\delta w_{ij}} \approx \frac{\Delta E}{\Delta net_i} x_j$ is used to make direct evaluation of the gradient. In this relation, $net_i = \sum_j w_{ij} x_j$ and $x_j = f(net_j)$ with f being the non-linear squashing function.

This leads to a weight update rule of the form: $\Delta w_{ij} \approx -\eta \frac{\Delta E}{\Delta net_i} x_j$

Therefore the inclusion of the MR III learning rule into an N neuron network in analog VLSI, additionally requires:

- An addressing module and wires routed to select and deliver the perturbation to each neuron.
- either one or N multiplication hardware to compute the term $\frac{\Delta E}{\Delta net_i} x_j$ in addition to the multiplication by the learning rate. If one multiplier is used then additional multiplexing hardware is required.
- An addressing module and wires routed to select and read the x_j terms.

Note that if greater training flexibility is required in the sense of off-chip access to the gradient values, then the states of the neurons (x_j) would need to be made available, which would require a multiplexing scheme or N chip pads.

2.1 Weight Perturbation

An alternative approach to node perturbation is "weight perturbation" where the gradient is approximated to a finite difference, and we show in this paper that gradient evaluation using "weight perturbation" is a cheaper solution, in both hardware and complexity, and can equally be used to train recurrent networks.

2.1.1 Gradient Evaluation using Weight Perturbation

The gradient with respect to the weight can simply be evaluated by the (forward difference) approximation

$$\frac{\partial E}{\partial w_{ij}} = \frac{E(w_{ij}+pert_{ij})-E(w_{ij})}{pert_{ij}} + O(pert_{ij})$$

The weight update rule then becomes:

$$\Delta w_{ij} \approx -\eta \frac{E(w_{ij}+pert_{ij})-E(w_{ij})}{pert_{ij}} \qquad (1)$$

where $E()$ is the total mean square error produced at the output of the network for a given pair of input and training patterns and a given value of the weights.

The order of the error of the finite difference approximation can be made $O(pert_{ij}^2)$ by using the central difference method, and the weight update rule becomes:

$$\Delta w_{ij} \approx -\eta \frac{E(w_{ij}+\frac{pert_{ij}}{2})-E(w_{ij}-\frac{pert_{ij}}{2})}{pert_{ij}} \qquad (2)$$

However, the number of forward relaxations of the network required is of the order N^3 rather than N^2 for the forward difference method. Thus either method can be selected on the basis of a speed/accuracy trade-off.

Note that the weight update hardware involves the evaluation of the error with perturbed and unperturbed weights and then the multiplication by a constant. This technique is ideal for analog VLSI implementation for the following reasons:

1. No bidirectional circuits for back-propagation are needed as the gradient $\frac{\delta E}{\delta w_{ij}}$ is approximated to $\frac{E_{pert}-E}{\Delta_{pert} w_{ij}}$ (where $\Delta_{pert} w_{ij}$ is the perturbation applied at weight w_{ij}). i.e. the hardware used for the operation of the network is used for the training.

2. Compared to node perturbation our technique does not require the two neuron addressing modules, routing and extra multiplication listed above.

3. There are no overheads in routing and addressing connections to every neuron as the same wires used to access the weights are used to deliver weight perturbations. Furthermore, node perturbation requires extra routing to access the output state of each neuron and extra multiplication hardware is needed for the $\frac{\Delta E}{\Delta net_i} x_j$ terms which is not the case with weight perturbation.

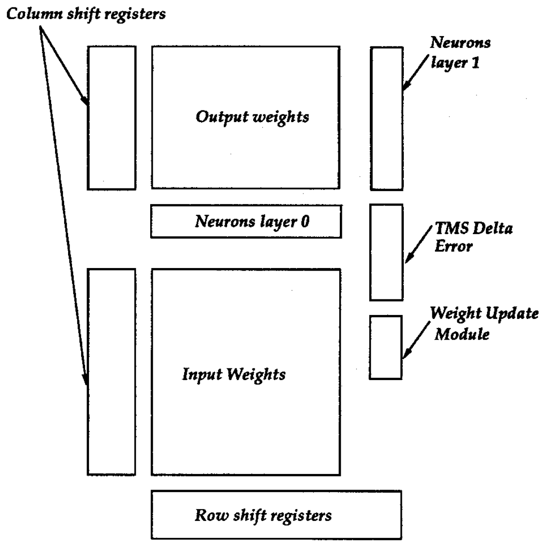

Figure 1. Implementation of Weight Perturbation

4. Finally, with weight perturbation, the approximated gradient values can be made available if needed at a rather low cost[1].

3. Architectures and System Level Design

In the previous section we presented "weight perturbation" (WP) and demonstrated it as cheaper for training than either back-propagation or MRIII. A typical analog sub-threshold architecture is shown in Figure 1. At the time of writing a chip is being manufactured that contains 7 input neurons, five hidden layer neurons and 3 output neurons (total of 50 synapses).

3.1 ANN State Representation

The architecture is based on a differential signals. The differential current/differntial voltage scheme is attractive because it offers:

- A reduction in the effects of corrolated noise on voltage and current levels.
- A direct interface to differential storage which offers a better immunity to charge decay.

Capacitor weight storage is permanently stored in RAM and periodically refreshed through a Digital to Analog

[1]If the mean square error is required off-chip then only one single extra pad is required. Otherwise, if approximated gradient values are to be calculated off-chip, then no extra chip area or pads are required as the output of the network would be accessible anyway.

Converter. Because we are not interested in true random access to the weights, weight decoding is performed by horizontal and vertical shift registers. Weight perturbations are applied simply, either by rewriting a perturbated digital value of the weight through the DAC or by direct charge or discharge of the weight storage capacitors.

The WP perturbation algorithm is implemented as follows:

A: Reset Column and Row weight decoding shift registers. Reset Total (epoch) Error

B: Save error on Error-Capacitors. Add Error to Total Error Capacitors.

C: Apply perturbation to current weight

D: Save Error-Change.

E: Update selected weight

F: Shift Row weight decoding shift register

G: If Row shift register has overflow, then shift Column weight decoding shift register. If Column weight decoding shift register has overflow then: if Total Error E convergence criteria then stop; else Goto A:

H Goto to B:

A synapse is made up of a weight store (two MOS gate capacitors), a multiplier, and a column/row select decoding scheme. It has two pairs of differential input voltages: one from a neuron and the other pair from the weight voltage on the two capacitors. a synapse outputs a differential current (I+ and I-) which facilitates summing at a neuron input.

A neuron has two subblocks: an impedance input stage and a squashing function as an output stage. The input to the neuron is a differential current (the I+ and I- from the synapse) and its output is a differential voltage.

4. Circuit Design

As a precursor to implementing a full network we had a 'test run' to check the operation of elemental neural network functions and validate our cad tools. To keep design costs to a minimum and for a reasonable turn-around time the "Tiny Chip" design offered through Orbit's Foresight Programme was selected[2]. The sub-circuits implemented were: Current divider; Current divider plus (p) mirror; Wide range transconductance amplifier; Four quadrant Gilbert multiplier; Neuron(I) with current dividers on input; Neuron(II) with multiplier on input; Weight storage capacitors with buffered output. The number of subcircuits was limited by the number of pads, only a small fraction of the available area being used.

[2]The technology used was n-well, single poly, double metal, $1.2\mu m$

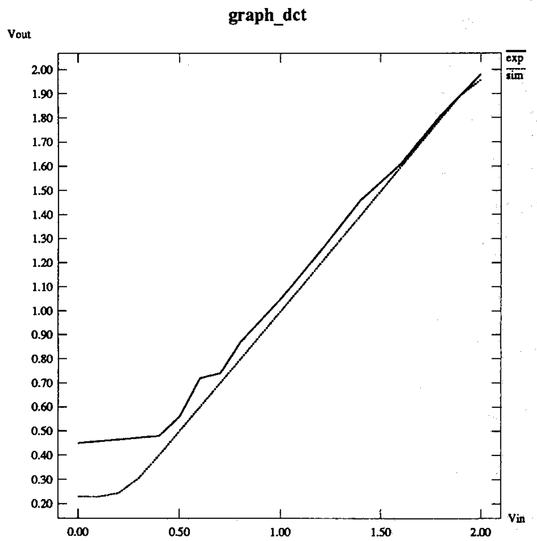

Figure 2. Synapse DC Transfer Function

The circuits were designed using Daisy's Ace and Vlab schematic capture and simulator software (using the Apex engine); laid out with Magic, the UCB silicon mask editor.

5. Simulation and Test Results

Due to limited space here, only the results for the synapse can be shown with a brief summary of the remainder.

5.1 Synapse

This comprises weight storage and multiplier elements. By using a drain-source shorted nfet whose gate is connected to the input of a unity gain buffer (a wide range amplifier with o/p connected to -ve i/p) the ability to store a 'weight' voltage was realised. The dc transfer function of this element is shown in figure 2, and its hold up time is plotted in figure 3. With a linear decay of only $3\mu V$ per second the prospect of a practical refresh period in a realistic weight matrix is very good.

There are two differential voltage inputs to the multiplier: Va and Vb. The results are for Vb set to ± 50 and $\pm 150mV$ with Va swept from $-200mV$ to $+200mV$. There are two outputs from the multiplier: a current source and a current sink. Figure 4 shows both the source and sink current outputs and these are in good agreement with the simulation results. For clarity simulations are

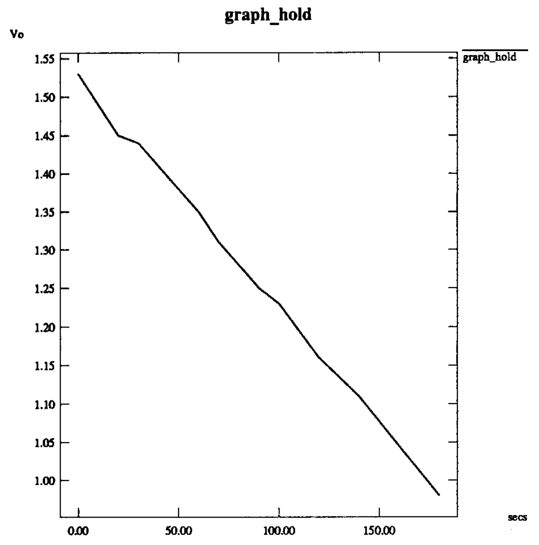

Figure 3. Synapse Decay Rate

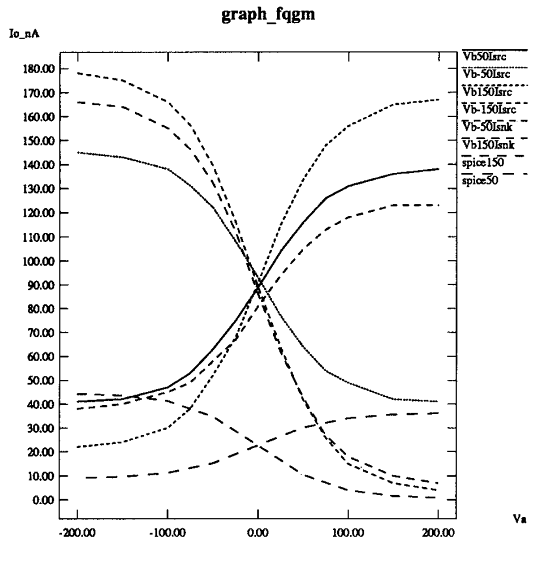

Figure 4. Four Quadrant Gilbert Multiplier

only shown for $Vb = +50and + 150mV$. The difference between simulation and measurement is again due to difference in threshold voltage. The current offsets at zero differential input voltage are small and are likely due to pad leakage.

It is intended that the input voltages to the multipliers (from weights and neurons) lie within their linear range ($\approx \pm 100mV$). This ensures that the only non-linearities are the neuron squashing functions. As the results show, the dynamic range is limited by the multiplier and current work aims to improve this.

5.2 Discussion of Results

All the tests proved satisfactory in that all of the functional elements operated correctly. The differences between predicted and measured performance seem to be due mainly to the variation in threshold voltage, and also pad current leakage in certain instances. The following points should be noted:-

1. The ratios of the current dividers were greater than expected but with very small offsets (14:1 instead of 11.2:1).
2. The output currents from the multiplier were larger than expected (approximately double) but again exhibited very small offsets.
3. Both neurons exhibited extreme sensitivity to gain and bias voltages which makes them susceptable to noise. More control over their operation is desirable (a new design has been included on the next chip).
4. The wide range trans-conductance amplifier exhibits the expected tanh curve though again there is an offset of about ten percent. Also the magnitude of the differential output current is much smaller than expected.
5. The weight storage transfer function shows good linearity over the range of interest (0.7 to 2 volts). The voltage decay rate is very small which will assist in the specification of any weight refresh circuitry.

References

[1] C. Mead. *Analog VLSI and Neural Systems.* Addison-Wesley Publishing Company, 1989.

[2] Bernard Widrow and Michael A. Lehr. 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. *Proceedings of the IEEE,* 78(9):1415–1442, 1990.