A VLSI Architecture for High-Performance, Low-Cost, On-chip Learning

Dan Hammerstrom
Adaptive Solutions, Inc.
Beaverton, Oregon

February 28, 1990

1 Introduction

The motivation for the architecture described in this paper was to develop inexpensive commercial hardware suitable for solving large, real world problems. Such an architecture must be systems oriented and flexible enough to execute any neural network algorithm, and work cooperatively with existing hardware and software. The early application of neural networks must proceed in conjunction with existing technologies, both hardware and software. Neurocomputation will succeed to the extent that it merges cleanly with existing computer structures. It is an enhancing, not a replacing technology.

Using state-of-the-art technology and innovative architectural techniques we can approach the speed and cost of analog systems while retaining much of the flexibility of large, general purpose parallel machines. We do not mean to imply that any point in the neurocomputer design space is necessarily superior to any other, rather, we have aimed at a particular set of applications and have made cost-performance trade-offs accordingly. Our vision is of an architecture that could be considered a general purpose, "microprocessor" of neurocomputing.

1.1 Architecture Goals

The architecture goals were adaptability, flexibility, low cost, speed, and ease of systems integration. These goals, as usual, often conflict. The X1[1] architecture is the result of a specific set of compromises that were made in attaining these goals. For a discussion of many of the complex issues facing neurocomputer designers, see "Artificial Neural Networks: Electronic Implementations" by Morgan [6].

1.2 Adaptability

Although many VLSI neural network implementations offer high-speed, few have on-chip learning capabilities. It is our belief that on-chip learning or adaptability is fundamental to neurocomputing. Hence, our primary design goal was to develop an adaptive system that learns efficiently on-chip, hence at high speed and without extra, off-chip hardware. Adaptability is essential for neural network research and for many applications.

1.2.1 Flexibility

The neural network field is in rapid transition, algorithms are changing, everything is in flux. Consequently, flexibility is a necessary characteristic of any general purpose neurocomputing engine.

[1] "X1" is our internal code name.

Flexibility is the ability of a system to utilize any neural network algorithm. The X1 succeeds in this goal by being programmable.

"Programmability" implies digital implementation. It is possible to build programmable analog systems, but cost-performance advantage is lost, and, given the current state of analog technology, there are manufacturing risks for mass production. Consequently, in the interest of flexibility, the first major decision was that the X1 would be an all digital implementation.

1.2.2 Low Cost

Low cost is a requirement to allow neurocomputers to proliferate into real world applications. The cost of implementation is kept low by using a medium that allows the mass manufacture of complex systems, and an architecture that reduces the cost within that medium.

For implementation, we have chosen CMOS. The intense competitive pressure in microprocessors and memories has forced CMOS to a level of cost per function and mass production unmatched by other technologies.

There are two ways to reduce the cost of the CMOS implementation. First, silicon area requirements must be made as small as possible. Here we begin to work against the other goals, and trade-offs are required. For example, the "multiplier" per synapse that is typical of analog computation, and which is where most of the speed of analog networks derives, can consume quite a bit of silicon area when precision requirements are increased and flexible weight update functionality is added to each weight site. By choosing a centralized digital update, we can reduce the "per synapse" area to that of a few small memory cells.

More centralized digital update allows significant reduction in silicon area requirements. And by reducing the precision of the computation to the lower limits required by the algorithms, the cost of digital update can be kept reasonable.

There are other motivations for using digital techniques. Bailey and Hammerstrom have shown that multiplexed communication allows for more cost-effective implementation in silicon of complex, high fan-out connectivity [1]. Such multiplexing is much easier with digital communication. Another advantage to digital is that it provides for the arithmetic precision requirements of many existing algorithms. Baker and Hammerstrom, and others, have shown that 8 and sometimes 16 bits of precision is necessary for the weight representation during neural network learning for certain algorithms such as Back-Propagation [2]. In lower cost CMOS analog technology, these levels of precision are difficult and expensive to attain. The X1 matches the precision requirements of the task of emulating a large range of neural network models, and it does so without excessive and expensive ALUs.

The second way to reduce CMOS cost is by using circuit redundancy to nullify inevitable manufacturing defects. Adaptive Systems is leveraging such techniques to keep die cost to a minimum.

1.2.3 Speed

Speed is increased by increasing concurrency in the computation. Neural networks are naturally suited for this, since they are massively parallel. Ideally one would like to utilize all the parallelism that is available, including the parallel computation of all synapses, as one sees in biological systems. This is one of the real strengths of analog based neurocomputation [5] [4]. However, the number of connections is an $O(n^2)$ cost, where n is the number of processing nodes. By using more centralized digital computation, we have traded off some speed for flexibility, precision, and lower cost per connection.

By making the individual processors as small as possible, it is possible to place a large number onto a single piece of silicon. By adding features that are customized to neural network computation it is possible to attain reasonable speed without sacrificing flexibility. Examples of these architecture features are discussed below. Selecting a leading edge silicon process allows high-density and high clock rate.

1.2.4 Systems Integration

Systems integration involves both hardware and software. A discussion of the integration of our architecture into a larger system is beyond the scope of this paper. However, by using programmable processors, multiplexed digital input and output, and by providing significant flexibility on chip, we have made the chip interface more like existing computational devices and thus have eased the task of integrating this architecture into a complete system. In addition, the use of a separate sequencer chip (the E1 in Figure 1) simplifies the interface of X1 neurocomputer arrays to existing microcomputer structures. This encapsulation is particularly useful in systems with a large number of X1 chips.

2 The X1 Architecture

The result of these various trade-offs is the X1 architecture. An X1 chip consists of a number of simple, digital signal processor like PNs (Processor Nodes), operating in an SIMD (Single Instruction stream, Multiple Data stream) configuration. Broadcast interconnect is used to create inexpensive, high-performance communication. The PN architecture is optimized for traditional neural network applications, but is general enough to implement any neural network algorithm (learning and non-learning) and many feature extraction computations, including classical digital signal processing, pattern recognition, and rule based processing (fuzzy and non-fuzzy).

2.1 Processor Array

The X1 system consists of a linear array of PNs. Each PN is a simple arithmetic processor with its own local memory. The array is sequenced by a single controller, thus each PN executes the same instruction at each clock, using SIMD processing. Each PN is connected to three global buses: InBus – the data input bus, PnCmd – the command bus, which indicates what operations a PN performs each clock, and OutBus – the output bus. The InBus and PnCmd buses are broadcast buses. All PNs see the same input data and command data. The OutBus can only serve one PN at a time for transmitting data. Several arbitration schemes are provided to control OutBus access. There is also an inter-PN bus between adjacent PNs that implements the OutBus arbitration algorithms and allows inter-PN data transfer. Figure 2 shows this basic structure. Chip boundaries across the array are arbitrary, since the user only sees a single array of PNs. Thus X1 systems are easily scalable.

Most neural networks can be created out of one or more basic two layer systems that perform a matrix vector multiply to produce an output vector, as shown in Figure 3. A number of factors distinguish this operation from traditional vector machines. These include lower precision arithmetic, non-linear output functions, mutual inhibition, and localized learning rules (matrix element modification).

This basic component, which we call a layer, takes a vector, multiplies it by a matrix of weights and creates a new vector (of typically different dimension). Any neural network structure can be created by this fundamental operation. Totally connected networks such as a Hopfield network use the output vector as the next input. Feed-forward networks can be thought of as several layers feeding each other successively.

A layer of CNs (Connection Nodes[2]) is typically emulated by a layer of PNs. More than one network element can be emulated by a single PN. In the basic X1 architecture a PN is usually allocated to each CN. But this may not always be the case. It is possible to assign multiple CNs to a PN, via time division multiplexing, or to spread a single CN across multiple PNs. A feedforward network usually emulates one layer at a time. Figure 4 shows the mapping of a two-layer network to a single layer of PNs. In a recursive network the outputs would feed back to the inputs.

[2]We use the term CN to indicate that this is an emulated network node. This distinction is required, since the correspondence between CNs and PNs is not always one to one.

SIMD PN execution is controlled by a separate sequencer chip which contains a writable control store, and a microsequencer. The microsequencer is capable of placing data and/or literals onto the InBus and for sequencing commands to the X1 array.

2.2 Interconnect Structure

Neural networks tend to have $O(n^2)$ connections, where n is the number of nodes. The most efficient multiplexed interconnect architecture for large fan-in and fan-out is broadcast. Consequently, broadcast structures were chosen to implement X1 input and output; n uses of a broadcast medium can "make" n^2 connections. Another advantage of broadcast is that it is simple. No complex routing networks are required. This saves control circuitry area and the need to program complex interconnect structures, thus reducing silicon area costs.

A disadvantage of simple broadcast interconnect occurs when less than total interconnect is required. In this case, not all listening nodes will be able to use the communicated data, and these PNs will be idle. Since X1 PNs are small and simple, and the interconnect structure inexpensive, a certain level of sparse connectivity is possible before the cost-performance loss becomes excessive. Future versions of X family architectures will utilize patented hierarchical and overlapped broadcast structures[3] to more efficiently handle sparse interconnect structures which will be increasingly common in neural network models.

The current X1 architecture allows for separate parallel input buses. Such an architecture could be used for a vision processing system where each bus would input data for different subregions of the visual field, allowing a significant increase in network input bandwidth. Examples are shown in Figure 1.

2.3 Processor Node Architecture

Each Processor Node has internal units connected via control signals and several data buses. See Figure 5. The InBus and PnCmd bus enter the PN through the Input Unit. In general the PN is horizontally microcoded:

1. *Input Unit:* The Input Unit decodes the PnCmd bus and routes it to the other units, and receives an 8 bit value from the InBus. It also contains a flag to allow conditional instruction execution of each PN. Although the input and output buses are 8 bits, the internal buses are 16 bits. The Input Unit can assemble two 8 bit quantities into a 16 bit value when needed.

2. *Logic-shifter:* The Logic-shifter contains both a shifter and a logic operation unit. Both operate on 16 bit quantities. These allow computed quantities to be manipulated by shifting and bit masking. There is also a 1's counter that sums the bits in the byte and whose output is used with 1 bit weights (8 per byte) to accumulate the AND of an 8 bit input and an 8 bit weight. This permits binary inputs and binary weights (8 per byte) – 8 are computed each clock.

3. *Register file:* The register file contains 32 16-bit registers for intermediate storage of constants such as learning rates and the PN ID.

4. *Multiplier:* The Multiplier unit contains a 9x16 bit, two's complement multiplier. The multiplier produces a 24 bit output. Using mode bits, the personality of the multiplier can be modified to multiply a positive 8 bit number by a signed 16 bit number, a signed 8 bit number by a signed 16 bit number, or a signed 16 bit number by a signed 16 bit number (this operation takes two clocks). There is a direct path from the multiplier to the adder that allows for the PN to be operated in *Vector Mode* where a simultaneous multiply and accumulate can occur each clock.

[3]Patent No. 4,796,199, additional patents pending.

5. *Adder:* The adder/subtractor takes 32 bit inputs and produces a two's complement 32 bit result. Adder overflow causes saturation to the largest positive or negative number, depending on the sign of the final result.

6. *Weight Address Generation:* The separate multiplier-to-adder bus allows the PN to be operated in a vector mode, which requires a regular stream of addresses to be generated for the weight memory. This unit contains its own adder for adding the contents of the stride register to the current weight index. As a result there is arbitrary striding through memory for those programs that have more complex data structures.

7. *Weight Memory:* Memory can be accessed in either 8 bit or 16 bit mode. There is also a hardware system, called the Virtual Zeros mechanism[4], which is used for the efficient representation of sparse connectivity. This is a simple set of bit mapped registers that allow groups of contiguous zeros in memory to be removed from real memory.

8. *Output Buffer:* The Output Buffer contains the arbitration logic for access to the Inter-PN bus and the OutBus. Both 16 and 8 bit values can be transmitted. 16 bit values require 2 clocks to transmit over the 8 bit OutBus.

   The OutBus arbitration and transmission mechanism operates separately from, but is synchronized with, the SIMD control to allow PNs to both transmit and do multiply-accumulation simultaneously. This capability is required when outputs of a layer are fed back as inputs to the next layer. This feature is called the Virtual PN[5]. There are several OutBus arbitration modes.

The X1 has three basic weight modes: 1 bit, 8 bits (7 mantissa+sign), and 16 bits (15 mantissa+sign). When the mantissa is zero, it is generally considered a null weight. Most algorithms require 8 bits, but there are some algorithms that need only single bit precision and some that require up to 16 bits. Two's complement representation is used throughout. Likewise, all arithmetic saturates to the maximum value on overflow, or minimum value on underflow. When bits are truncated, a form of bit jamming can be selected that eliminates error bias.

Values transmitted onto and off of the chip can be either 8 or 16 bits. The preferred mode is 8 bits, since for most neural network algorithms that is sufficient. The 16 bit mode is provided for those applications, such as digital signal processing or the use of 16 bit weights, that need more precision. The use of 8 bit output allows thresholding functions (such as the sigmoid) to be implemented with a simple table lookup.

The inter-PN bus is also used to perform a multi-PN maximization function where the PNs cooperatively determine, using a decentralized algorithm[6], which PN has the maximum value. This function allows for the efficient implementation of Winner-Take-All networks. In addition, multiple, arbitrary Winner-Take-All regions can be defined and operated in parallel. This "Max" function operates independently of chip boundaries. The basic algorithm operates on 1 bit at a time and hence can go to arbitrary precision.

3 Physical Implementation

The first implementation of the X family architecture is the X1 chip. A chip plan for a single PN is shown in Figure 6. A chip has 64 PNs, each PN has 4,096 bytes of weight storage. Thus, one chip can store the weights for 262,144 (256K) 8 bit connections, 131,072 (128K) 16 bit connections, or 2,097,152 (2M) 1 bit connections. Vector mode allows each PN to perform a multiply-accumulate per clock, which means the computation of one connection per clock per PN for 8 and 16 bit

[4]Patents pending.
[5]Patents pending.
[6]Patents pending.

weights. At 25 Mhz, the maximum performance in non-learning mode is 1.6 billion connections computed per second per chip, assuming all PNs are utilized. For 1 bit weights, the maximum performance is 12.8 billion connections computed per second.

In back-propagation learning mode, the additional computations required for weight update reduce the speed per input vector by about a factor of 6 relative to the simple feedforward computations of the testing mode. Thus, one chip can do back-propagation learning at about 260 million connection updates per second, assuming all PNs are utilized. This number assumes a back-propagation network with 16 bit weights and 8 bit activation values.

4 Performance

The entire NetTalk network (203 input nodes, we used 64 hidden nodes, and 21 output nodes) can be put onto a single chip, and can be trained using the standard back-propagation algorithm over the approximately 76,800[7] vector training set reported by Sejnowski and Rosenberg [7] in just under 8 seconds.

A Discrete Fourier Transform of 128 points (real coefficients only, 16 bit values), as would be required in a speech system, takes 10 microseconds.

Best match on multiple features, as one would do, for example, in Batchelor's algorithm [3], is also possible. Arbitrary distance functions can be used. A single chip can store a network with 2,048 prototypes, each with 128 inputs of 8 bit weight precision per element. A best match (assuming a Euclidean metric) on all prototypes can be done in 170 microseconds. This is roughly 5,800 128 element vectors per second. These algorithms typically require normalized input. Normalization of a 128 input vector with 8 bit precision per element can be done in about 10 microseconds per vector.

If 1 bit weight precision is used (as one might find in a visual processing system of 1 bit pixels) then 16,384 prototypes can be stored. Best match processing time in this case is 240 microseconds. In 1 bit weight mode, the chip is evaluating 68 million 128 input prototypes per second.

5 Conclusions

The X1 meets our architecture goals of adaptability, flexibility, low cost, speed, and ease of systems integration. We have met our flexibility goals in that we can emulate any known neural network algorithm, including those requiring on-chip learning. In addition, the X1 allows discrete Fourier transforms to be performed on chip as well as segmentation, neural network feature extraction and rule-based decision processing. Although the X1 has a larger than typical die size, the ability to build a complete neurocomputing system of this power from only two chips: a single X1 and an E1 allows us to meet our cost goals.

The X1 is well suited for use in a variety of research and industrial applications. It represents the first member of an evolving and upward compatible line of neurocomputer architectures.

References

[1] Jim Bailey and Dan Hammerstrom. Why vlsi implementations of associative vlcns require connection multiplexing. In *Proceedings of the 1988 International Conference on Neural Networks*, June 1988.

[2] Tom Baker and Dan Hammerstrom. Characterization of artificial neural network algorithms. In *1989 International IEEE Symposium on Circuits and Systems*, pages 78–81, September 1989.

[7]This number assumes 15 passes through the original 1,024 word corpus of informal speech, with an average of 5 letters per word.

[3] Bruce G. Batchelor, editor. *Pattern Recognition: Ideas in Practice*, chapter 4. Plenum Press, 1976.

[4] Hans P. Graf, Lawrence D. Jackel, and Wayne E. Hubbard. Vlsi implementation of a neural network model. *IEEE Computer*, 21(3):41–49, 1988.

[5] Mark Holler, Simon Tam, Hernan Castro, and Ronald Benson. An electrically trainable artificial neural network (etann) with 10,240 floating gate synapses. In *Proceedings of the IJCNN*, 1989.

[6] Nelson Morgan, editor. *Artificial Neural Networks: Electronic Implementations.* Computer Society Press Technology Series and Computer Society Press of the IEEE, Washington, D.C., 1990.

[7] T. Sejnowski and C. Rosenberg. Nettalk: A parallel network that learns to read aloud. Technical Report JHU/EECS–86/01, The Johns Hopkins University Electrical Engineering and Computer Science Department, 1986.

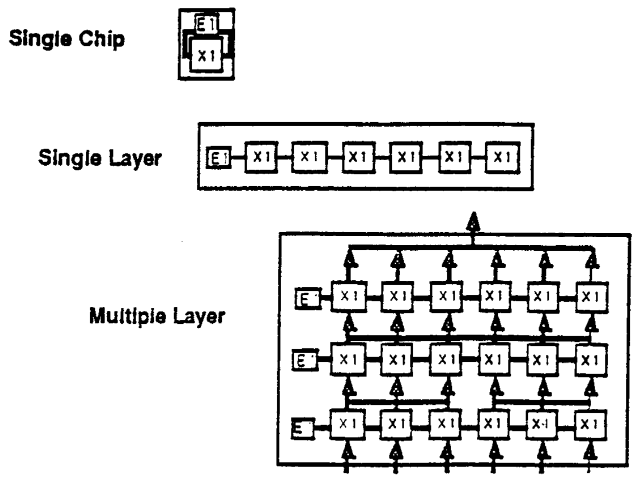

Figure 1. X1 SYSTEM EXAMPLES

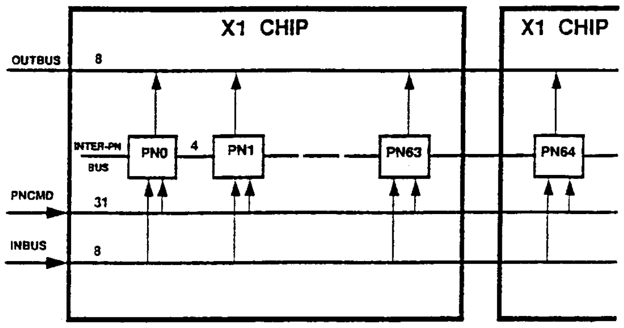

Figure 2. X1 ARRAY

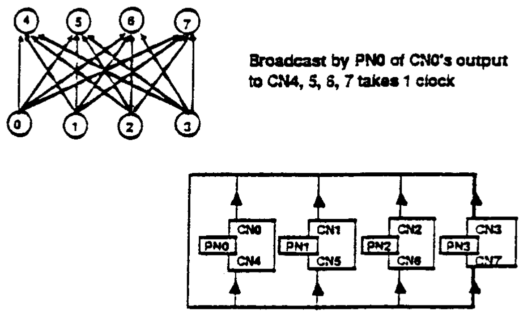
Broadcast by PN0 of CN0's output to CN4, 5, 6, 7 takes 1 clock
CN0
CN4
PN0
CN1
CN5
PN1
CN2
CN6
PN2
CN3
CN7
PN3

Figure 4. PN CONNECTIVITY

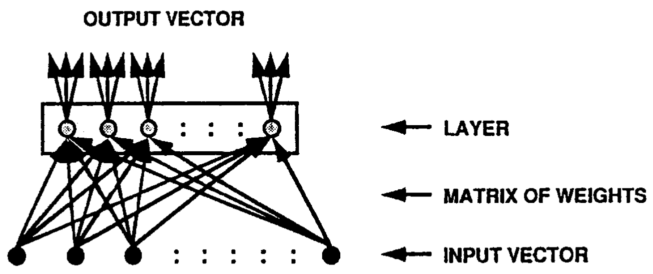

Figure 3. NEURAL NETWORK COMPONENT

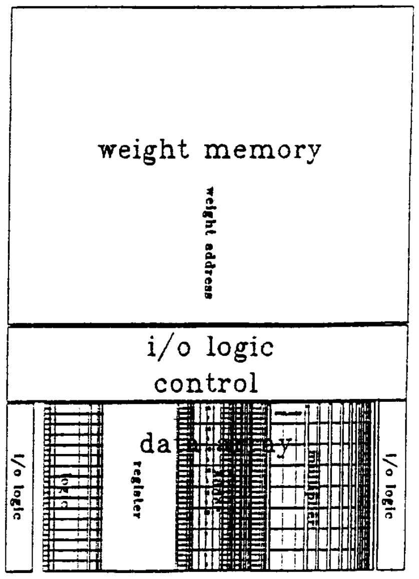

Figure 6. PN CHIP PLAN

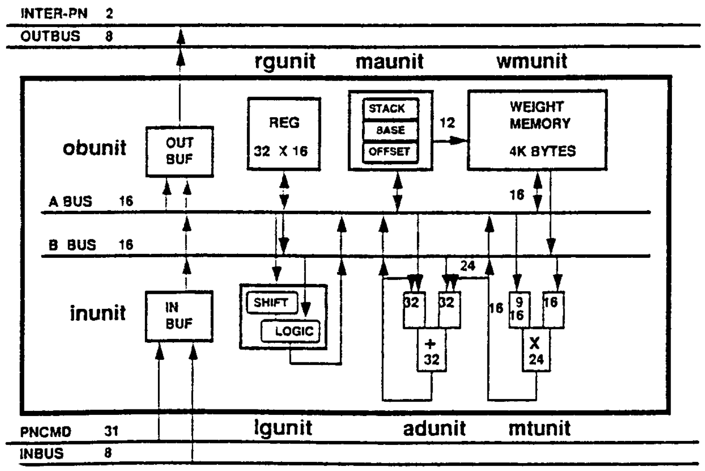

Figure 5. PROCESSOR NODE