Developments in Digital VLSI Design for Artificial Neural Networks

Nelson Morgan[1], Krste Asanovic[1,2]
Brian Kingsbury[1,2], John Wawrzynek[2]

TR-90-065

December, 1990

Abstract

Artificial Neural Networks (ANNs) have been heralded as a form of massive parallelism that may significantly advance the state of the art in machine intelligence and perception. While these expectations may or may not be realistic, this class of algorithms has already been useful for difficult problems in signal processing and pattern recognition over the last 25 years. However, for extension to a wider class of problems, a key requirement is the parallel hardware implementation of such systems, since ANN implementation on conventional Von Neumann machines is often prohibitively slow. While the ANN mainstream has focused on analog VLSI ANNs, some projects have shown the potential of a fully digital approach. We report here on progress in developing a methodology for digital ANN design, including a new object-oriented CAD interface, and a set of ANN-specific library cells. A new measure for efficiency of silicon ANNs is also described.

[1]International Computer Science Institute, Berkeley, California
[2]University of California at Berkeley, Berkeley, California

INTRODUCTION

Artificial Neural Networks (ANNs) are systems that, like the real networks for which they are named, consist of a number of elements that perform computations on the weighted outputs of one another. Such a system could also be called "connectionist" [1], because information is represented in the pattern and strength of connections between elements. Analogies between such systems and biological networks have probably been exaggerated in the last few years. However, experience over the last 25 years has confirmed that even fairly simple systems of this type can be useful in such problem areas as signal processing, pattern recognition, and artificial intelligence [2][3].

One of the key advantages of such systems is the apparent match to parallel implementation. By providing a family of algorithms that are inherently parallel, ANNs offer the promise of simpler development of powerful silicon engines for machine intelligence. However, researchers have thus far achieved limited results. Special-purpose designs have performed well [4][5], but have required a lengthy design process. The high performance of these designs is probably due to their special-purpose character; they are not programmable, and very little silicon is used for control.

This tradeoff between specificity of function and programmability is a fact of life in VLSI design. The key to high performance in custom ICs has been the rapid development of associated CAD techniques, so that the limits of the technology can be pushed without prohibitive design times. This has been particularly apparent in signal processing, where semi-automatic tools have generated high performance designs for speech and image processing [6][7][8].

Unfortunately, the corresponding tools for ANN-CAD are still in their infancy. High performance ANNs require major design efforts, and minor changes to algorithms cannot typically be incorporated quickly in the circuit design. Part of this inflexibility is due to the fact that most ANN chips are analog and experimental [9]. In time the art may mature, and powerful CAD tools may be developed for analog ANN circuits. However, analog approaches have a number of limitations that are currently problematic for prospective ANN system designs:

1) Resolution/accuracy - In principle, analog circuits can represent continuous variables (i.e., have infinite resolution), and ANN algorithms are generally modeled on biological mechanisms that are known to function with inaccurate components. However, practical circuit restrictions limit the smallest resolution with which values can be represented or discriminated from one another. For instance, the Intel ETANN chip, which uses "floating gate" mechanisms for storage of connection strengths, can only store weights that are one of 16 possible values, over a dynamic range of about 100.

   Furthermore, many of the more popular ANN algorithms poorly model biology, and frequently require a wide dynamic range in order to converge to useful solutions. In particular, stochastic gradient algorithms such as back-propagation have been shown to require 12-16 bits of range, or roughly four orders of magnitude between the largest possible weight values and the smallest weight change [10][11]. This class of ANN algorithms is actually the one that is most commonly used today, and analog circuits have been unsuccessful at providing the resolution required for convergence of the learning algorithm.

2) Scaling to smaller geometries - While functions can be calculated directly by device physics in clever analog designs, these computations tend to be more critically dependent on the circuit size than they are for scalable CMOS digital designs. In particular, thermal noise power is roughly inversely proportional to the length unit (λ) of the process so that extremely small analog ANN circuits may be so noisy as to be unreliable for realistic algorithms. Thus, as minimum circuit sizes scale to submicron dimensions, performance improvements should be more evident in the digital world than in the analog one.

3) On-chip Communication - The ultimate limitation in planar silicon designs is the connectivity between elements. For the particular case of ANNs, a number of analyses have shown that large systems require communication multiplexing. Analog signals can also be multiplexed, but at the cost of

further tricky design.

4) Off-chip Communication - It is tricky to interconnect high-speed analog signals at the board level, due to the effects of radiated noise, crosstalk, power supply isolation problems, etc. Digital communication in a large system is currently reliable up to the Gbit/second rate.

5) Virtual Neurons - A simple probabilistic analysis [11] shows that large silicon connectionist systems may require each physical computational element to implement many more than one "neural" unit. This computational multiplexing may be implemented with analog components, but much care must be taken to reduce the effects of crosstalk, and a large silicon area is required for the digital control circuitry.

6) Inflexibility - For most cases, minor algorithmic changes (e.g., the unit nonlinearity) force a major analog circuit redesign. For digital circuits, such changes can frequently be accommodated by simply generating a different logic array.

7) System Integration - With the exception of circuits integrated with transducers, most computational circuits must work with standard digital components such as semiconductor memories, disks, uniprocessor CPUs, etc. In this case the "real world" is digital, and interface for system operation and testing is much more convenient with a digital part.

The last point is in practice one of the most important. ANNs are subject to Amdahl's Law[1], so that speed improvements from a lightning-fast analog "neuron" may not be visible if the remaining subsystems are implemented by relatively slow digital components. Therefore, a simpler approach would be to design the entire system using fast digital implementations that will work well together, both for "neural" and "non-neural" parts. We believe that such an approach is a viable solution to the ANN system design problem, and that custom IC design can provide overall system performance that at least approaches the throughput of a clever analog system design.

These considerations have led to the development of a design methodology for digital ANNs that is described in the remainder of this paper, including a preliminary design example. We first describe some new developments in VLSI design at Berkeley that are relevant to our goals.

OBJECT-ORIENTED CUSTOM IC DESIGN

Ideally, the task of implementing an custom IC design of an ANN would consist solely of the preparation of a behavioral description for input to a silicon compiler. Silicon compilation is an active research area, but presently human intervention is necessary in the behavioral to physical translation process if competitive designs are to be achieved. This labor can represent a considerable expenditure in design time.

As device densities increase we can expect far greater emphasis to be placed on the reuse of existing subsystems. The VLSI designer's task has already become less overall chip design and more implementation of library cells catering to a wide variety of users. At the highest level, relatively naive users will require modules equivalent to application specific silicon compilers. For example, a high level library might be capable of generating complete ANNs parameterized by unit type and interconnection pattern. High level systems designers will view VLSI as simply an advance in packaging technology and will expect to be able to combine predesigned modules as easily as they currently combine chips on a board. Gate array and standard cell technologies are currently popular solutions in this area. Moving towards more

[1]Amdahl's Law : the performance improvement to be gained from using some faster mode of execution is limited by the fraction of time the faster mode can be used (as stated by Patterson and Hennesy in [12]).

aggressive designs, VLSI designers will require much more malleable libraries that can be tuned to particular design objectives such as ANNs. This level of tuning will require rich interfaces to the library modules as many interdependent parameters may need to be considered. At this level of design, the user should have the tools to hand design the required part. On the other hand, the library must be sufficiently flexible and easy to use to render these hand designs unnecessary most of the time. However, the design of library cells requires a uniform mechanism for composing circuit primitives and pre-existing library modules to form new library modules.

These requirements for flexibility, reusability, and extensibility mirror the more general software requirements that have generated widespread interest in object-oriented programming languages such as Smalltalk or Eiffel, and for object-oriented extensions of traditional languages, such as C++. The key observation is that a modern object-oriented language can implement CAD specific objects that are as easy to manipulate as primitive objects in a special purpose procedural layout language. CAD objects are merely data structures with associated operations, and map naturally into the object-oriented paradigm.

An initial experimental system, BOSS, has been developed as part of a graduate course at U.C. Berkeley [13]. BOSS consists of C++ classes built on top of the Berkeley Octtools database [14][15]. The Berkeley Octtools database provides persistent storage for design data and is widely used in academia and industry. Many tools have been developed that communicate through this database. C++ was chosen as it is a relatively mature and stable language, and provides a natural interface to the C code that comprises most of the existing CAD software [16]. The classes in BOSS can be split into those that provide a wrapper around the objects supported by the Octtools database, and higher level abstractions to support VLSI layout.

For example, one higher level abstraction is a two dimensional array of cell instances. Operations defined on this array allow rows and columns to be populated with other instances. Further operations allow connections to be routed between elements within the array a row or column at a time. During layout generation the array elements are interrogated to determine bounding box information. Many popular VLSI structures, such as RAMs, ROMs, and PLAs, map naturally into this kind of array abstraction. By factoring out the layout code common to all such structures, we can substantially reduce the development time of new structures.

BOSS also provides some very simple routing functions. These are intended to be used under programmer control to implement small local wiring connections in known wiring patterns. Such fairly regular local communications dominate the wiring in most designs, and giving the programmer explicit control saves execution time and potentially produces better results than a large scale router. This example illustrates a general theme in this approach. The use of tool functions needs to be built into library cell definitions. Hence, tools should also be provided as classes operating on CAD objects, not as monolithic entities. This has the advantage of simplifying the design of the tool, since no user interface is required and what would previously have been implemented as a single monolithic tool can now be provided as many smaller, more specialized tools.

Several library cells have been laid out using BOSS, including an ANN datapath described later in this paper. These experiments have convinced us of the value of this technique. BOSS was only intended as a prototype system, and our experience has suggested a number of major modifications. We are therefore currently developing a successor system, OctC++, which will be used for our future ANN VLSI designs [17]. While BOSS was designed as a single module to speed implementation, OctC++ is being split into more manageable chunks. The lowest layer of OctC++ will consist of C++ wrappers around the Octtools database. This code will help isolate the higher layers from future changes in the design database. Higher level layers will implement the abstractions present in BOSS, such as cell arrays and wiring pragmas. BOSS is fairly conservative in its use of the novel features of C++, while OctC++ attempts to further capitalize on inheritance, dynamic typing, and operator overloading to ease the library design process.

So far the discussion has centered on layout generation. However, we believe the object-oriented approach can be profitably applied to other areas of VLSI design.

Object oriented techniques were originally developed to aid in the writing of simulations [18], which are among the more time consuming parts of VLSI design. C++ is already a popular language in which to perform simulation work, and we are making extensive use of the language in our architectural simulations. In particular, the ability to modify the behavior of arithmetic operations without changing algorithmic code is especially useful for ANN design, due to the range of arithmetic precision required for different

variables in an ANN (e.g., 12-16 bit for connection strengths and 1-8 bits for unit outputs).

We are also investigating ways to integrate functional simulation, circuit simulation, and chip testing. We already perform functional simulation in C++, in effect creating a C++ object to model our hardware element. The interface to this C++ object defines the required interface to the hardware system. If we provide circuit simulators as library tools operating on layout objects, it becomes possible to implement the same C++ interface by interacting with generated layout through the simulator code. Hence the same algorithmic code used to define the architecture can be used to test an implementation. The same interface can also be implemented using libraries that communicate with a chip tester. The same algorithmic code can then be used to functionally test fabricated chips. In C++ we would define an abstract base class representing the hardware interface. The designer would then inherit this interface in three subclasses corresponding to the three different implementation of the interface.

Perhaps the greatest advantage of this object-oriented approach is the common control through the same general-purpose language of many different libraries and objects. There are no restrictions on how these elements can be combined with each other and with non-CAD specific libraries. For example, with C++ it is possible to use the Stanford InterViews library to provide an object-oriented interface to the X windowing system for graphical interfaces.

In summary, CAD design is increasingly a software art. Object oriented design developed for software management is directly applicable to CAD design resulting in the same benefits of increased reuse, enhanced flexibility, and straightforward extensibility. We are currently in the process of building a design system with these goals in mind. Later in this paper we describe our experience with the preliminary system (BOSS) for ANN hardware design.

APPLICATION: SPEECH RECOGNITION

To demonstrate the applicability of these techniques to a problem of interest, it is necessary to take a small detour into a machine perception application of interest to us at ICSI. In our work on ANN applications, we have developed a phoneme-based, speaker-dependent continuous speech recognition system [19]. The system utilizes a layered ANN to generate emission probabilities for a Hidden Markov Model (HMM) recognizer. We have shown that this method is an effective way to smoothly estimate joint densities with a number of training samples that is insufficient for simple histogram-based techniques. The ANN is capable of performing statistical pattern recognition over the undersampled pattern space without many restrictive simplifying assumptions. The ANN can also combine multiple sources of evidence, such as multiple features and contextual windows, in a straightforward and efficient manner. Initial experiments indicate that this method compares favorably with conventional HMM speech recognition systems [20].

In this system, continuous features (such as spectra) are extracted from speech input and passed to a vector quantizer that maps the input speech frame into one of a set of prototype vectors or features. The ANN is fed a sequence of vector quantized frames that provide a sliding window into the input speech stream. The ANN uses this contextual information to recognize phonemes and/or generate the probability of each phoneme given the input.

Each vector quantized frame is represented using unary encoding. That is, there is a binary input neuron for each possible feature value, only one of which can be active at a time. For the networks described in [19], the vector quantizer selects one from 132 feature values and the ANN is fed a window of 9 frames. This gives $132 \times 9 = 1188$ input layer neurons, of which only 9 will be active in a given pattern. The output layer consists of 50-64 neurons, corresponding to the number of phonemes to be recognized. The best experimental results were obtained without hidden units, and so the input layer is directly and fully connected to the output layer. Error back-propagation training is used [21][22].

The network algorithms are summarized in the following equations. Input neuron i is connected to output neuron j by a weight w_{ij}. The output of neuron j is given by

$$o_j = f(s_j) \qquad (1)$$

where

$$s_j = \sum_i o_i w_{ij} \qquad (2)$$

and

$$f(x) = \frac{1}{1+e^{-x}} \qquad (3)$$

Bias values are treated as an extra weight connected to an always active input neuron.

Input values are either 1 or 0, and for each feature within each frame only a single input is active. We can split weight memory into 9 banks, one per frame. The sum can then be efficiently computed as

$$s_j = \sum_k W(k, v(k)) \qquad (4)$$

where $W(k,v)$ selects weight v in weight bank k, and $v(k)$ is the feature value for input frame k. We require only 9 additions to sum the unit bias and the 9 connection weights for each vector-quantized feature. No multiplies are required, which simplifies the hardware; however, the connections for each input pattern are sparse (mostly zeros), which complicates the design.

The training algorithm derives a weight update value Δw_{ij} using a cross entropy error criterion [23]

$$\Delta w_{ij} = -\alpha(o_j - d_j)o_i \qquad (5)$$

where d_j is the desired output value for neuron j, and α is the learning rate. Note that o_i is 0 for inactive inputs so the corresponding weights w_{ij} need no updating. For the active inputs, the value of Δw_{ij} is $-\alpha(o_j - d_j)$ which is independent of i. With no performance degradation, α can be restricted to negative powers of 2. This replaces the learning constant multiplication with an arithmetic shift. For the single-feature case, we require 10 additions for each neuron to perform weight updates, 9 for the weights and one for the bias. Note that the difference between output and target can be simply calculated by a PLA, since the target is always a 1 or a 0.

Reducing arithmetic precision is important as it decreases weight storage requirements, as well as reducing datapath circuitry. The results of our simulations indicate that in this application a weight precision of 12-16 bits is sufficient for learning using back-propagation. Output values require 6-8 bits. These findings are in agreement with those discovered by researchers working with back propagation in other application areas [11]. The systems we describe use 12 bit weights and 6 bit output values. Additionally, we used only 6 bits for the weight increment Δw_{ij}, corresponding to the least significant bits of the stored weight, used only the most significant 6 weight bits for the forward connection strengths.

A further modification was made to the algorithms to simplify the hardware. The 9 input frames presented to the network are grouped into three groups of three corresponding to past, present, and future frames. Within each group, the three weights corresponding to a given feature vector are tied together so that any update affects all three. This modification was originally added to improve generalization in the speech recognition system, but has the additional advantage of reducing the weight storage required in an implementation by a factor of three. The three frames within each group share a single bank of weight memory. The total weight storage per neuron is 3×132×12=4752 bits (not counting the bias value).

These modifications were simulated on a Sun workstation, and for a German language test set of 200 sentences, performance matched our previously reported results within a few tenths of a percent. Thus, it appears that a fixed-point algorithm with the required simplifications for efficient VLSI implementation works essentially as well as the original floating-point version. This algorithm should map efficiently to the cells that we are developing for ANN implementation.

DESIGN EXAMPLE: A SIMPLE DATAPATH

We have been studying the design of building blocks for the speech recognition ANN described above [24]. In the course of this study we have compared our procedural approach to VLSI design with DPP, a datapath generation tool from a Berkeley CAD management utility called Lager [25]. Lager acts as a supervisory wrapper for a collection of design tools that use some of the features of the Oct database. We made this comparison by designing a simple datapath using Lager, and then doing a functionally

equivalent design using BOSS and the Octtools VLSI CAD toolset. The datapath is the simplest possible implementation of the arithmetic elements of a single-layer perceptron with binary inputs, the ANN construct used in the speech-recognition research described above. The architecture of the datapath is shown in Figure 1. Both designs were fabricated using the MOSIS 2.0 micron n-well process and then tested.

The sequence of steps used to produce a layout under the Lager design system was straightforward and simple. First, a specification of the datapath logic without the logic needed to do two's complement saturating addition was developed, simulations were carried out to ensure the correctness of the specification, and a layout of the datapath was generated. Next, saturation logic was specified, simulated, and implemented, and a complete datapath was assembled from the two blocks. Then, the control logic for the datapath was specified, simulated and implemented, and the datapath and control logic were brought together to form the chip core. The chip itself was then assembled from the chip core and a specification of the pads. Before being sent out for fabrication, the chip design was simulated to ensure its correctness.

Figure 2 is a die photograph of an 8-bit Lager datapath design, fabricated as a 2 micron MOSIS Tiny Chip. The large block in the center of the chip is the datapath. The saturation control logic is above the left side of the datapath, and the control logic is below the datapath on its right side. The chip is functional at 25 MHz (the design specification was 20 MHz). We found that the design proceeded rapidly, because there was no need for hand layout of cells and because Lager provides a mechanism similar to the UNIX "make" utility that helps the VLSI designer manage changes to the design.

The design process using BOSS and the Octtools was slower and more involved, primarily because all datapath cells layouts had to be done by hand. First, the datapath was broken down into a collection of leaf cells such as single-bit pipeline latches and two-to-one multiplexers. Next, all leaf cell layouts were done, and the leaf cell designs were simulated to ensure functional correctness[2]. Extensive SPICE simulations of the saturating adder cells were done to ensure that the adder would operate at 20MHz. Next, C++ code using the BOSS library of routines was written to assemble the leaf cells into circuit blocks such as N-bit saturating adders. A code sample from one of these programs is shown in Appendix A. The correctness of the layout generators and the leaf cells they use was verified by simulating several instances of each generated circuit block. Then, another program was written to assemble the datapath from the smaller circuit blocks. The datapath generated by this program was simulated, and then a controller for it was synthesized using the Mississippi State standard cell library. Finally, the datapath and controller were simulated together, placed in a pad frame, and routed to generate the final chip layout.

Figure 3 is a layout illustration of the BOSS/Octtools datapath design. The large block in the center of the chip is the datapath, and the small block above it is the controller. The chip is functional at 20MHz. While the design of this chip took much longer than the Lager design, we feel that the time was well spent: the datapath design is about half the size and consumes about half the power of the Lager design. Furthermore, the BOSS/Octtools datapath is more easily tiled than the DPP design because the BOSS/Octtools design does not require an external adder saturation logic block.

While this comparison is useful, its results are not surprising. The BOSS/Octtools design is essentially a full-custom design in which knowledge about the architecture of the chip could be used to guide the low-level implementation. Thus, we were able to pitch-match all datapath cells and mirror them so that they could share power and ground lines. Knowing that all cells would communicate through short connections to other datapath cells, and that buffering could be provided as necessary to drive long lines, we were able to use relatively small transistors for the cell layouts. Because we knew we would need a saturating adder, we were able to build saturation logic into our adder design. The DPP library cells were designed for general use, so it is not surprising that our custom design uses much less silicon while operating at roughly the same speed.

It seems, then, that we face a classic dilemma: to achieve high performance, one must do a time-consuming custom design; to produce a design quickly, one must use a silicon compilation system and sacrifice some circuit performance. We believe that our object-oriented approach to VLSI CAD has the potential to alleviate this problem. The BOSS/Octtools design produced more than just a datapath chip; it also produced parameterized block generators for a saturating adder, two-to-one multiplexer, transparent

---

[2]Cell layouts within the (symbolic) Oct policy contain implicit connectivity information, which facilitates the direct simulation of layout. The Octtools set includes a simulator for this purpose.

latch, and pipeline latch. Furthermore, many of the cells and almost all of the code used to generate the saturating adder could be used to generate a standard adder, and the routines used for block generation are easily modified to generate other circuit blocks. Thus, although the first few chips we do will require a significant design effort in the layout of cells and construction of block generators, later designs will proceed faster as we accumulate a collection of tools customized for the designs we create. Ultimately, we expect that design time for our chips will approach what is available through the use of more traditional silicon compilation tools, while maintaining a level of performance in our designs equivalent to what is available through full-custom design.

DISCUSSION: MAPPING AN ANN TO SILICON

The example shown above, while falling short of a complete ANN design, is illustrative of the kind of considerations required for a digital VLSI approach. Most research on silicon ANNs has focused on the fast implementation of dot products, either for connecting all units or for connecting one layer of units to another in the case of feedforward nets. The dot product of interest is the connection weighting of equation (2). One simple way to obtain fast dot products is by using a commercial processor that has been optimized for single cycle multiply-accumulate operations, such as a DSP chip. Multiple DSPs have been assembled in systems [26][27] that yield significantly higher performance than uniprocessors for the ANN algorithms of interest. Part of the attraction of such systems is their programmability, particularly important in a development environment.

A DSP chip, however, uses a significant portion of its area to support features that are not useful for ANN algorithms. A custom IC optimized for a limited number of ANN sequences would require a tiny amount of control logic and use a simplified bus structure compared to a commercial DSP. These differences, along with the more modest data word width for a customized chip, will yield throughput improvements of as much as two orders of magnitude for the custom chip, using comparable technologies. Furthermore, in many instances, as in the speech example above, algorithmic variations can often at least partially remove the requirement for explicit multiplications.

The performance of these approaches can be normalized by using the silicon area expressed in a technology-independent way using the die area in units of of λ^2. This is shown in the first column of Table I, for a number of implementations of forward propagation for a fully connected layered net, which is essentially a dot product. Note that in the table we have defined a Connection Per Second as CPS, and that the unit performance normalized by the scalable area is called a COnnection Rate Density (CORD) for this algorithm. The Adaptive Solutions chip, which is optimized for 16-bit dot products, shows much of the expected improvement [28]. A further improvement is predicted for a dedicated pipeline approach [29] using the cells described earlier in this paper. Finally, an analog chip recently developed by Intel, the ETANN, gives the best performance, using relatively inaccurate weights. However, the improvement over the digital custom approach is slight.

Using a related normalized performance measure, the Connection Update Rate Density (CURD), these chips can also be compared for their ability to learn. The ETANN chip does not perform on-chip learning, but if a similar technology were used to compute weight changes, the tunneling process used to program connection strengths would limit the area-normalized performance to about .03 CURDs. This is only an order of magnitude better than the fully programmable DSP chip, and quite a bit worse than the custom digital IC. Of course, other analog weight change mechanisms can be quite a bit faster, so this analysis does not prove that "digital is better." However, the real point is that fast design techniques to generate extremely dedicated hardware, be it digital or analog, can bring enormous performance benefits.

While the CURD and CORD measures (which roughly correspond to the standard AT performance measure for general VLSI designs) appears to be a useful way of comparing silicon ANNs, they still have a number of limitations. Firstly, they still are not measures of the technology-independent efficiency of the design, since the measures favor small geometries (which would have a faster connection or connection update rate). This can be accounted for by a further scaling by the value of λ in microns, yielding NCORDS and NCURDS for the normalized versions. Note that these final measures essentially correspond to $\frac{1}{AT^2}$, since the computation times are roughly inversely proportional to the minimum line widths. For the cases of Table I, this final scaling would only affect the digital custom design row, which would be multiplied by 2, since our MOSIS implementations are done in 2-micron CMOS, as opposed to the 1

micron processes used in the other cases.

A more fundamental limitation to these measures is the usual problem of condensing system performance into a single number. The CORD and CURD ratings give no insight into the utility of the system, its testability, how smoothly the ANN can be integrated with other components, etc. There is no easy solution to this objection; the reader must, as always, not be content with any succinct performance metric, and use his or her intelligence to consider the larger context.

Networks with sparse connectivity are implemented with poor efficiency on dot-product engines, be they analog or digital. This case can be handled in custom ANN design by sending addresses as well as data (where the latter is required if the input is not binary). In the implementation of our speech system, for instance, each speech pattern would consist of addresses of weights to be summed, corresponding to "on" inputs determined by vector-quantized speech features. Such an application benefits from having a moderate number of distributed memories, as opposed to the extremes of a weight vector per processor (fully parallelized), or one central bank (as in a uniprocessor implementation). This trade-off can be optimized for each new application.

SUMMARY

We have proposed that silicon ANNs can be flexibly and efficiently implemented with custom digital ICs designed with object-oriented CAD. We have also described the initial application of these techniques to the design of a datapath for use in a target problem in speech recognition. Finally, we have evaluated the performance of several existing and proposed circuit implementations of common ANN algorithms, using a measure that shows the relative effectiveness of silicon usage for the application. This result shows that digital implementations of ANNs can be efficient, and furthermore that they can realize functions that are difficult for an analog implementation (such as gradient descent using a wide dynamic range).

ACKNOWLEDGEMENTS

Thanks to James Beck for his advice and comments along the way. Mark Beardslee was responsible for the implementation of BOSS, and Andrea Casotto provided major assistance for Octtools use; their contributions are greatly appreciated. We also acknowledge the long-term aid from a number of people in the Lager group, as well as others from other participants in the Berkeley CAD project. The National Science Foundation has provided explicit support for this project with Grant No. MIP-8922354, and also with Graduate Fellowship support for Brian Kingsbury. John Wawrzynek received support from the National Science Foundation through the PYI award, MIP-8958568. Last but not least, we gratefully acknowledge the support of the International Computer Science Institute.

REFERENCES

[1] J.A. Feldman, D.H. Ballard, "Connectionist models and their properties," *Cognitive Science 6*, 3, pp. 205-254, 1982.

[2] S.S. Viglione, 1970, "Applications of pattern recognition technology," in *Adaptive Learning and Pattern Recognition Systems*, J.M. Mendel and K.S.Fu, Eds., New York, Academic Press, 1970, pp. 115-161.

[3] A. Gevins, and N. Morgan, 1988. "Applications of Neural Network (NN) Signal Processing in Brain Research," IEEE Transactions on Acoustics, Speech, and Signal Processing, July, 1988.

[4] D.B. Schwartz, R.E. Howard, and W.E. Hubbard, "A Programmable Neural Network Chip," *Journal of Solid-State Circuits*, Vol. 24, No. 2, April 1989, pp.313-319

[5] M. Holler, S. Tam, H. Castro, and R. Benson, "An Electrically Trainable Artificial Neural Network (ETANN) with 10240 "Floating Gate" Synapses," International Joint Conference on Neural Networks, Washington D.C., pp. II-191-196, 1989

[6] H. Murveit and R.W. Brodersen, 1987, "An Integrated-Circuit-Based Speech Recognition System,"

IEEE Trans. Acoustics Speech and Signal Processing, Vol. ASSP-34, No 6 (December).

[7] J.M. Rabaey, R.W. Brodersen, A. Stoelzle, D. Chen, S. Narayanaswamy, R. Yu, P. Schrupp, H. Murveit, and A. Santos, "A Large-Vocabulary Real-Time Continuous-Speech Recognition System," in *VLSI Signal Processing III*, edited by R.W. Brodersen and H.S. Moscovitz, IEEE Press, New York, NY, 1988

[8] P. Ruetz and R.W. Brodersen, "A Custom Chip Set for Real-Time Image Processing," ICASSP-86, v2, pp. 801-804

[9] F. Faggin and C. Mead, "VLSI Implementation of Neural Networks," in *An Introduction to Neural and Electronic Networks*, ed. S. Zornetzer, J.Davis, and C. Lau, Academic Press, San Diego, 1990

[10]N. Morgan, *Artificial Neural Networks: Electronic Implementations* (editor), Computer Society Press Technology Series Computer Society Press of the IEEE Washington, D.C., In Press

[11] T. Baker, and D. Hammerstrom, 1988, "Modifications to Artificial Neural Networks Models for Digital Hardware Implementation," Oregon Graduate Center Technical Report No. CS/E 88-035.

[12] D.A. Patterson and J.L. Hennessy, *Computer Architecture A Quantitative Approach*, Morgan Kaufmann Publishers, San Mateo, 1990

[13] M. Beardslee, Internal Document, EECS Dept., U.C. Berkeley, 1990

[14] R.L. Spickelmier, "Oct Tools Distribution 4.0", August 1990, UC Berkeley

[15] D.S. Harrison, P. Moore, R.L. Spickelmier, and A.R. Newton, "Data Management and Graphics Editing in te Berkeley Design Environment," Proc. IEEE Intl. Conf. on CAD, pp. 24-27, 1986

[16] M. A. Ellis and B. Stroustrup, *The Annotated C++ Reference Manual*, Addison Wesley, Reading Mass.,1990

[17] K. Asanovic, "OctC++: A C++ Interface to the Oct Database," ICSI Technical Report, In Prep

[18] O.J. Dahl, K. Nygaard, "Simula - An Algol-based Simulation Language," CACM 9;9 pp. 671-678, Sept. 1986.

[19] N. Morgan, H. Bourlard, 1990, "Continuous Speech Recognition Using Multilayer Perceptrons with Hidden Markov Models," Proc. IEEE Intl. Conf. on Acoustics, Speech, & Signal Processing, pp. 413-416, Albuquerque, New Mexico, 1990.

[20] N. Morgan, C. Wooters, H. Bourlard, and M. Cohen, "Continuous Speech Recognition on the Resource Management Database using Connectionist Probability Estimation," ICSI Technical report TR-090-044, also to be published in proceedings of ICSLP-90, Kobe, Japan

[21] P.J. Werbos, 1974, "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences," Ph.D. thesis, Dept. of Applied Mathematics, Harvard University, 1974

[22] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, 1986. "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing*. vol. 1: Foundations, Ed. D.E. Rumelhart and J.L. McClelland, MIT Press, 1986.

[23] G. Hinton, "Connectionist Learning Procedures," *Artificial Intelligence* Volume 40, Number 1, pp. 143-150, 1989

[24] B. Kingsbury, "Library cells for Digital Artificial Neural Networks," ICSI Technical Report, In Prep

[25] J.M., Rabaey, S.P. Pope, and R.W. Brodersen, "An Integrated Automated Layout Generation System

for DSP Circuits," IEEE Trans. on CAD, Vol. CAD-4, No.3, pp.285-296, July 1985

[26] N. Morgan, J. Beck, P. Kohn, J. Bilmes, E. Allman, and J. Beer, "The RAP: a Ring Array Processor for Layered Network Calculations," *Proc. of Intl. Conf. on Application Specific Array Processors*, pp. 296-308. IEEE Computer Society Press, Princeton, N.J., 1990

[27] A. Iwata, Y. Yoshida, S. Matsuda, Y. Sato, and N. Suzumura, "An Artificial Neural Network Accelerator using General Purpose Floating Point Digital Signal Processors," *Proceeding IJCNN 1989*, pp. II-171-175

[28] D. Hammerstrom, "A VLSI Architecture for High-Performance, Low-Cost, On-chip Learning," International Joint Conference on Neural Networks, San Diego, 1990

[29] K. Asanovic, B. Kingsbury, N. Morgan, J. Wawrzynek, "A Highly Pipelined Architecture for Neural Network Training," Proceeedings of 1990 IFIP Workshop on Silicon Architectures for Neural Nets, In Prep

Appendix A: Sample of BOSS Layout Generation Code

```
bossPipelatch(int height)
{
// Define output of pipeline latch generator

 BossFacet facet = BossFacet(form("pipelatch.%d",height),"symbolic");

// Set up array of single-bit pipeline latches

 BossInstArray pipelatch(height,1);
 pipelatch.populate_all(facet,LATCH_LOC,"physical","latch");
 pipelatch.offsetBB_all(LATCH_BB_TOP,LATCH_BB_BOT,
                LATCH_BB_LEFT,LATCH_BB_RIGHT);

// Mirror every other latch in Y so that power and ground lines are shared

 for(int row=0;row<height;row++)
  if(row%2 == height%2)
   pipelatch.transform_row(OCT_MIRROR_Y, row);

// Place pipeline latches in array

 pipelatch.place_all();

// Specify nets (logical connections used for simulation) between latches

 pipelatch.connect_col(0,"clk",BossNet(facet,"clk"));
 pipelatch.connect_col(0,"clkn",BossNet(facet,"clkn"));

 // Do reflection-dependent nets

 int vflag = 0;
 for (row=height-1;row>0;row--) {
   if (vflag) {
     BossNet Vdd(facet,form("Vdd<%d>",row/2));          // Create net
     Vdd.add_term(pipelatch.elmt(row,0).term("Vdd"));   // Attach terminals
     Vdd.add_term(pipelatch.elmt(row-1,0).term("Vdd")); // to net
   }
   else {
     BossNet GND(facet,form("GND<%d>",row/2));
     GND.add_term(pipelatch.elmt(row,0).term("GND"));
     GND.add_term(pipelatch.elmt(row-1,0).term("GND"));
   }
   vflag = 1-vflag;
 }

// Specify formal terminals (the interface to other circuit blocks)

 pipelatch.promote_all(facet,"in1","in1");          // Convert terminal "in"
                                                    // of each latch to a
                                                    // formal terminal "in<row>"

 pipelatch.promote_all(facet,"in2","in2");
 pipelatch.promote_all(facet,"out1","out1");
 pipelatch.promote_all(facet,"out2","out2");
 pipelatch.promote_all(facet,"Feed1","Feed1");
```

```
pipelatch.promote_all(facet,"Feed2","Feed2");
BossFormalTerm(pipelatch.elmt(height-1,0).term("clk"),"clk.t");      // Convert terminal "clk.t"
                                                                     // of the top latch to the
                                                                     // formal terminal "clk"

BossFormalTerm(pipelatch.elmt(0,0).term("clk"),"clk.b");
BossFormalTerm(pipelatch.elmt(height-1,0).term("clkn"),"clkn.t");
BossFormalTerm(pipelatch.elmt(0,0).term("clkn"),"clkn.b");
BossFormalTerm(pipelatch.elmt(height-1,0).term("Vdd"),
        form("Vdd<%d>",height/2));

// Do reflection-dependent formal terminals

vflag = 0;
for (row=height-1;row>0;row--) {
  if (vflag)
    BossFormalTerm(pipelatch.elmt(row,0).term("Vdd"),
            form("Vdd<%d>",row/2));
  else
    BossFormalTerm(pipelatch.elmt(row,0).term("GND"),
            form("GND<%d>",row/2));
  vflag = 1-vflag;
}
if (vflag)
  BossFormalTerm(pipelatch.elmt(0,0).term("Vdd"),"Vdd<0>");
else
  BossFormalTerm(pipelatch.elmt(0,0).term("GND"),"GND<0>");
}
```

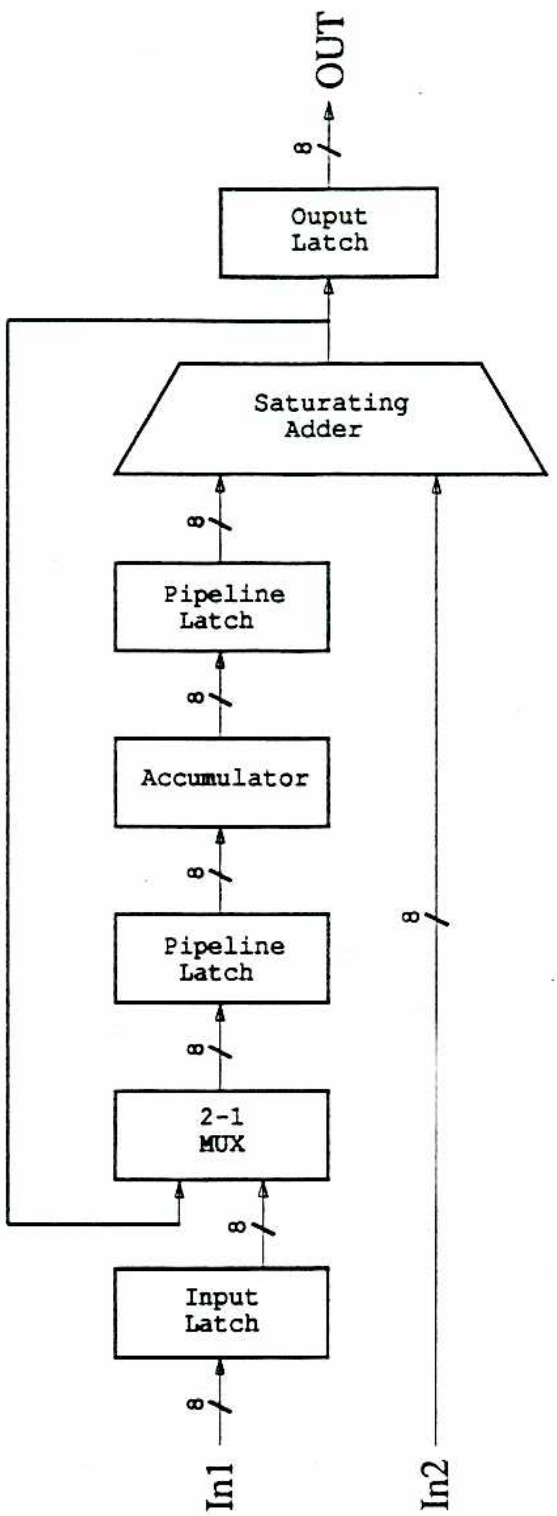

Figure 1: This is the architecture of the datapath described in our example. All parts of the datapath are standard datapath elements, except for the adder. Instead of overflowing, the adder will produce the largest possible output and instead of underflowing it will produce the smallest possible output.

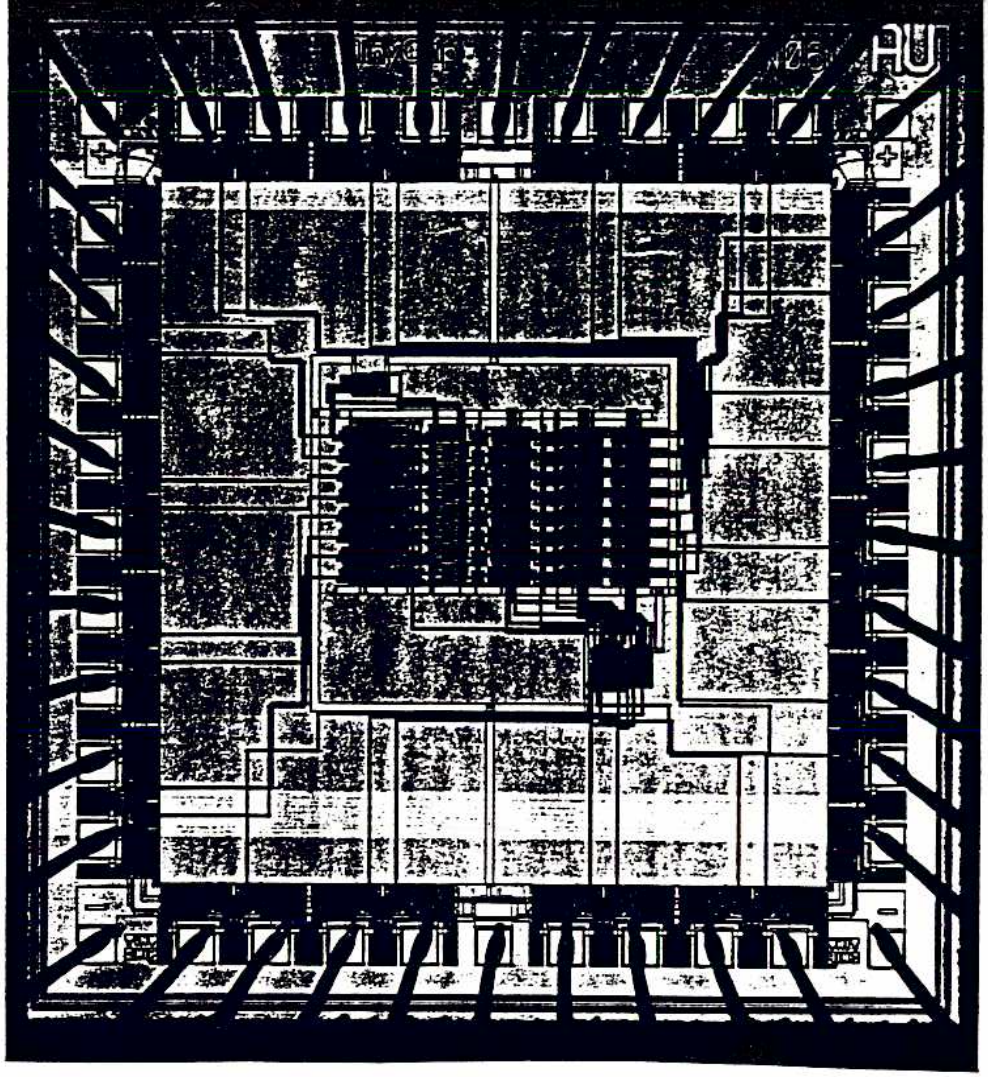

Figure 2: Simple Datapath--DPP Design

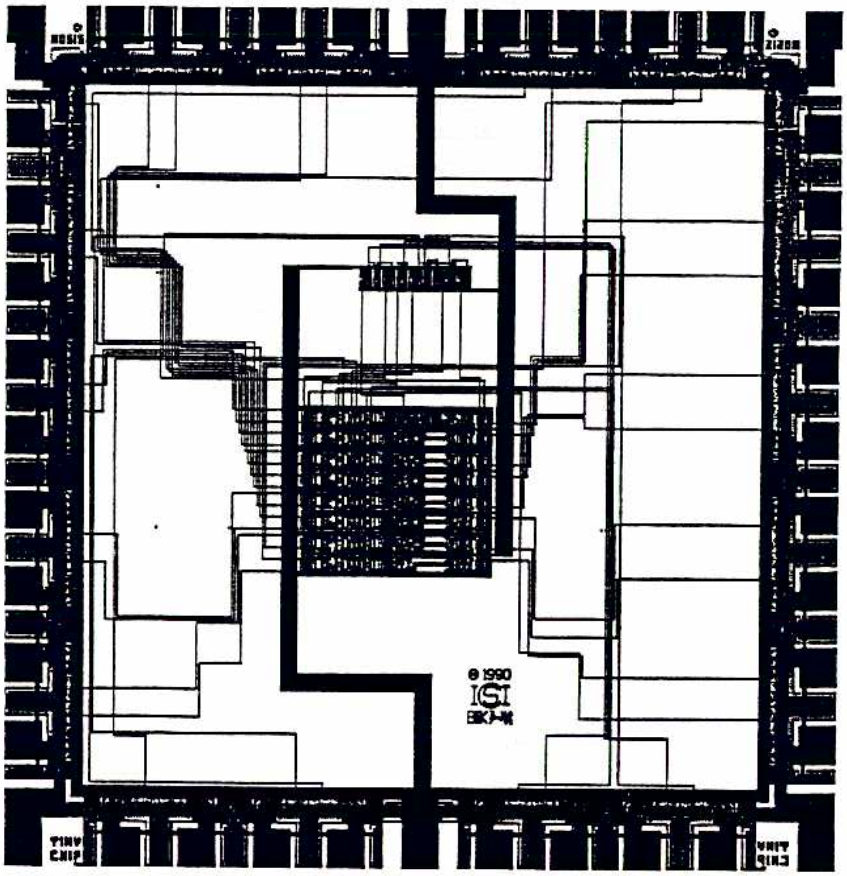

Figure 3: Simple Datapath--BOSS/Octtools Design

Table I: Connections Per Second / Normalized Silicon Area

	Forward	Full Learning
TI TMS320C30 (digital)	$\frac{1.6\times10^{7}CPS}{8\times10^{8}\lambda^{2}} = 0.02\ CORDs$ (floating pt weights)	$\frac{4\times10^{6}CPS}{8\times10^{8}\lambda^{2}} = 0.005\ CURDs$
Adaptive Solutions (digital)	$\frac{1.6\times10^{9}CPS}{1.6\times10^{9}\lambda^{2}} = 1\ CORD$ (16 bit weights)	$\frac{2.6\times10^{8}CPS}{1.6\times10^{9}\lambda^{2}} = 0.2\ CURDs$
Projected approach (digital)	$\frac{2\times10^{8}CPS}{10^{7}\lambda^{2}} = 20\ CORDs$ (12-16 bit weights)	$\frac{2\times10^{8}CPS}{2\times10^{7}\lambda^{2}} = 10\ CURDs$
ETANN (analog) (floating gate)	$\frac{10^{10}CPS}{3\times10^{8}\lambda^{2}} = 33\ CORDs$ (7 bits weight range)	speed of host (very slow) (tunneling process would limit to .03 *CURDs*)

Note: the areas for the 2nd and 3rd chips are estimates, based on assumed components and technologies