Digital Neural Network Implementations

James B. Burr
Department of Electrical Engineering
Stanford University
Stanford, Ca. 94305
burr@mojave.stanford.edu

January 2, 1995

Abstract

This chapter gives an overview of existing digital VLSI implementations and discusses techniques for implementing high performance, high capacity digital neural nets. It presents a set of techniques for estimating chip area, performance, and power consumption in the early stages of design to facilitate architectural exploration. It shows how technology scaling rules can be included in the estimation process. It presents a set of basic building blocks useful in implementing digital networks. It then uses the estimation techniques to predict capacity and performance of a variety of digital architectures. Finally, it discusses implementation strategies for very large networks.

1 Introduction

Neural network applications suitable for implementation in VLSI cover a wide spectrum, from dedicated feedforward nets for real time feature detection to general purpose engines for exploring learning algorithms. The DARPA Neural Network Study [76] contains a good discussion of the range of applications together with a method of classifying them which we use in this chapter.

The fastest analog neurochips are almost six orders of magnitude faster than an average workstation or PC, and three orders of magnitude faster than the fastest supercomputer. The fastest digital neurochip is about as fast as a supercomputer. Even so, many of the potential applications demand far more computational power and synaptic storage capacity than has been implemented so far. As progress is made toward implementation of larger, higher performance networks, and as technology scales further into the submicron regime, power dissipation will become a dominant constraint, and effective power analysis and optimization will be an important aspect of design.

There is always a tradeoff in VLSI implementation between performance and flexibility. Both are needed in neural nets. Mature applications with well characterized algorithms will continue to seek higher performance as learning algorithm research continues to suggest new architectures.

This chapter discusses both existing implementations and some of the challenges facing future implementations. Section 2 proposes an informal classification scheme for neurochips. Section 3 describes some of the more recent digital VLSI implementations. A table at the end of section 3 gives a quick overview of implementations to date. Section 4 discusses architectural tools which can be leveraged in implementing high performance, high capacity networks. Section 5 describes some low level basic building blocks useful in digital implementations. Section 6 describes techniques for estimating area, performance, and power of a desired architecture, and shows how technology scaling rules can be included in the estimation process. Section 7 discusses techniques for maximizing performance on a constrained power budget. Section 8 discusses the impact of technology scaling on future neurochip architectures. Section 9 discusses a chip we have been working on at Stanford. Section 10 discusses obstacles and opportunities to implementing large digital networks.

2 Classifying VLSI implementations

Yutaka Akiyama's PhD thesis [4] has a comprehensive section on neurochip classification. Since its publication, the number and variety of neurochips has proliferated. We can group these chips into major categories depending on whether they use analog or digital computation, how synaptic storage is implemented, whether they have some form of on-chip learning, and whether they are standalone or are meant to be connected to other chips to implement larger networks.

The most common analog architecture is a fully connected mesh of N^2 synaptic processors supporting N neurons. This architecture can perform feedforward computations at very high rates, typically converging in a few microseconds, and is well suited to Hopfield nets [39] and associative memories [26, 25]. For example, ATT's associative memory chip [25] has 256 neurons and 65536 synapses. Assuming 10 iterations through the network for convergence, this chip has a feedforward computation rate of around 500 GCPS (billion connections per second).

Another analog architecture used in the Caltech retina chip [71] is a lattice of neural processors with nearest neighbor connections. This architecture can also compute at very high rates. The chip has 2304 neurons, each connected to 6 neighbors. Assuming convergence in a few microseconds, this chip has a feedforward computation rate of around 10 GCPS.

Digital chips tend to be organized as a 1 dimensional systolic array [47, 46, 45]. They generally have lower feedforward computation rates than analog chips; values reported range from 3 MCPS [22] to 1.28 GCPS [28].

Analog implementations usually compute the inner product as an analog current sum, but differ in how the weights are stored. Techniques reported include resistors [25, 26, 71], CCDs

[2, 15, 65], capacitors [19, 48, 56], and floating gate EEPROMs [38].

Digital implementations use either parallel, bit-serial, or stochastic arithmetic. Some have one processor for each synapse; most have one processor per neuron or even one processor for many neurons. Synaptic weights are either stored in shift registers, latches, or memory. Memory storage alternatives include one-, two-, or three-transistor dynamic RAM, or four- or six-transistor static RAM.

Most digital implementations so far include some form of on-chip learning; most analog implementations do not.

A few hybrid chips have been reported which combine analog and digital techniques [7, 55, 24]. One common approach is to store weights digitally and to compute the inner product as a current sum.

Most chips that learn store weights digitally. One notable exception is a purely analog Boltzmann chip [9], which uses its learning algorithm to refresh capacitive weights. This chip is implemented in 1.0μ 2-poly 2-metal CMOS. It has 125 neurons and 10000 capacitive synapses. Each synapse implements the Boltzmann weight update rule [34, 1] with 42 transistors, and measures $200\lambda \times 200\lambda$. Network convergence in response to a single input takes 5 microseconds, including annealing, which is implemented as a damped oscillation on a reference voltage. Assuming a 50 step anneal schedule, the equivalent digital feedforward computation rate would be about 100 GCPS.

3 Existing digital implementations

3.1 TI's NETSIM

NETSIM [22] was one of the first purely digital architectures reported. It consists of two types of chips: a "solution engine" chip and a "communication handler" chip. Neural computation is done in the solution engine; neural activations are routed through the communication handler. The solution engine requires external memory to store weight values. A solution engine, a communication handler, a local microprocessor, and memory are packaged on a single NETSIM card. A system consists of a number of NETSIM cards in a 3-D array interfaced to a general purpose computer.

The solution engine performs an 8x8 bit multiply-accumulate in 250ns, so the performance is 4 MCPS.

These chips do not appear to have been implemented, but the architectural ideas appear in various forms in later chips. The idea of pipelined off chip weights is especially appealing for very large networks. Also, considerable attention was given in the design to multichip implementations.

3.2 Duranton and Sirat's digital neurochip

Duranton and Sirat's proposed chip [18] is a fully digital architecture which stores 16 bit synaptic weights in an on-chip RAM. The architecture supports on-chip learning, but exports the sigmoid function. It uses an interesting bit-serial technique to form the inner product.

On each cycle, one bit of each activation is anded with the weights to form a set of partial products which are reduced to a single partial product in a tree of adders. Activations are 8 bits, so it takes 8 cycles to form $x_i = \Sigma w_{ij} x_j$ for a single i. The weights are fetched in parallel out of the RAM, so the RAM access time can be 1/8 the processor cycle time.

The performance estimations are given for 1.6 micron CMOS. The paper states that a new x_i is computed every 2 μsec. This implies a clock frequency of 25 MHz, and with 32 16-bit synaptic weights in parallel on every cycle, 800 MCPS. This chip also does not appear to have been implemented.

3.3 Hirai et al's digital neuro-chip

Yuzo Hirai and colleagues [36] fabricated a digital neurochip using a 1.2 micron CMOS gate array. The chip implements six neurons and 84 6-bit synapses using a variant of pulse-stream arithmetic (see section 5.7. Rather than use a chopping clock as in [57], or a comparator and pseudo-random number generator as in STONN [78], TInMANN [16], or DNNA [42], Hirai et al supply each neuron a separate, asynchronous clock to reduce pulse collisions. Each neuron requires 1 msec to evaluate its inputs. This "time constant" depends only on the number of bits in the synapses and not on the number of synapses. This implies a performance of 84 KCPS. Other pulse-stream implementations achieve better performance by allowing the network dynamics to overlap the neural computation.

3.4 Ouali and Saucier's Neuro-ASIC

Ouali and Saucier [59] have implemented a single neuron on a chip in 2.0 micron CMOS which runs at 20 MHz. The chip is a "cascadable neuron processor" that includes a local memory for storing synaptic weights and activation function parameters, a multiplier, an adder/subtractor, a controller, and input, output, and state registers for interfacing to a multichip network. The paper does not specify precision or performance, though the chip has been fabricated as a MOSIS TinyChip [73].

3.5 Neural Semiconductor's DNNA

Tomlinson et al [42] have implemented a scalable neural net architecture (DNNA: digital neural network architecture) based on stochastic pulse trains. They have implemented two chips. The SU3232 contains 1024 synaptic elements arranged in a 32×32 matrix. The NU32

chip implements 32 neurons. A fully connected network with $32N$ neurons requires N^2 synapse chips.

Each synapse includes a separate stochastic pulse train generator. Each synaptic pulse stream is anded with an activation output stream to produce a synaptic product. Synaptic products are then wire-or'ed to produce activation input streams.

The neural activation function is implemented by anding excitatory and inhibitory activation input streams. Because the wire-or operation is naturally saturating, no extra sigmoid unit is needed to implement a nonlinear activation function.

A two-chip set can implement 32 neurons and 1024 connections. k bit accuracy in the activations requires 2^k cycles. The chips run at 25 MHz, and can process 100K activations per second at 8 bit resolution. This implies a feedforward computation rate of 100 MCPS, (200MCPS if the excitatory and inhibitory parts of each synapse are counted separately).

3.6 North Carolina State's STONN

Wike et al [78] have proposed a 100K transistor CMOS Hopfield network (STONN: stochastic neural net) using stochastic logic and bit-level pipelining. They leverage the recurrent nature of Hopfield nets to overlap the neural computation with network relaxation so that by the kth iteration the neural activation has been computed to a precision of $\log_2(k)$ bits. DNNA could do this as well.

Unlike the DNNA chip, which generates stochastic samples for N^2 weights on every cycle, STONN stores weights in an on-chip shift register and generates stochastic samples of N weights per cycle. This technique achieves much higher synaptic storage density than the DNNA approach, especially if the shift register weight store is implemented as a RAM.

A weight is converted to a pulse stream by comparing it to a sequence of random numbers, using the carry out of the most significant bit of a bit-level pipelined comparator to generate the pulses. The synaptic pulse stream is "multiplied" by an input activation stream by anding the two streams. At this point, rather than wire-or the modulated input activations as in DNNA, the modulated input activation increments or decrements a counter containing the accumulating output activation. The chip has one activation comparator which generates a single stochastic sample of one of the accumulating output activations each cycle.

Weights are 7 bits; comparators are 6 bits, activation accumulators are 10 bits. The weight store is implemented using a 6-transistor dynamic shift register. The chip has 20 neurons, and 100 7 bit weights per neuron. It is projected to run between 10 and 25 MHz (it has been layed out but not fabricated). At 10 MHz, one chip can achieve 200 MCPS; at 25 MHz, 500 MCPS.

Each iteration only involves one stochastic sample of weights and activations. This type of optimization network converges over time to a stable state. STONN takes N cycles to compute the outputs of N neurons. This is the same amount of time required by a parallel implementation with one processor per neuron. The stochastic nature of the computation is

much more efficient. The parallel computation is too exact. That is, one stochastic sample from one neuron can be generated each cycle. Parallel algorithms also generate one sample per cycle, but to higher precision and requiring more hardware.

Reduced precision parallel computation can require comparable hardware. The multiplier-accumulator in STONN consists of an and gate, a 6-bit comparator, a random number generator, and a 10-bit up-down counter. The multiplier in the Stanford Boltzmann chip (see section 9) is a 5-bit full adder (shared Booth encoding reduces 5 partial products to 3) and a 10-bit carry-save accumulator. The 5-bit full adder matches STONN's 6-bit comparator. The accumulator matches STONN's random number generator. The carry-save accumulator matches STONN's up-down counter.

Given bit-level pipelining, STONN should run closer to 100 MHz than 10 MHz in 2.0 micron CMOS, unless power consumption is a problem, implying 5 GCPS. It should also be able to accommodate around 64 neurons and 32K weights using one-transistor DRAM.

3.7 North Carolina State's TInMANN

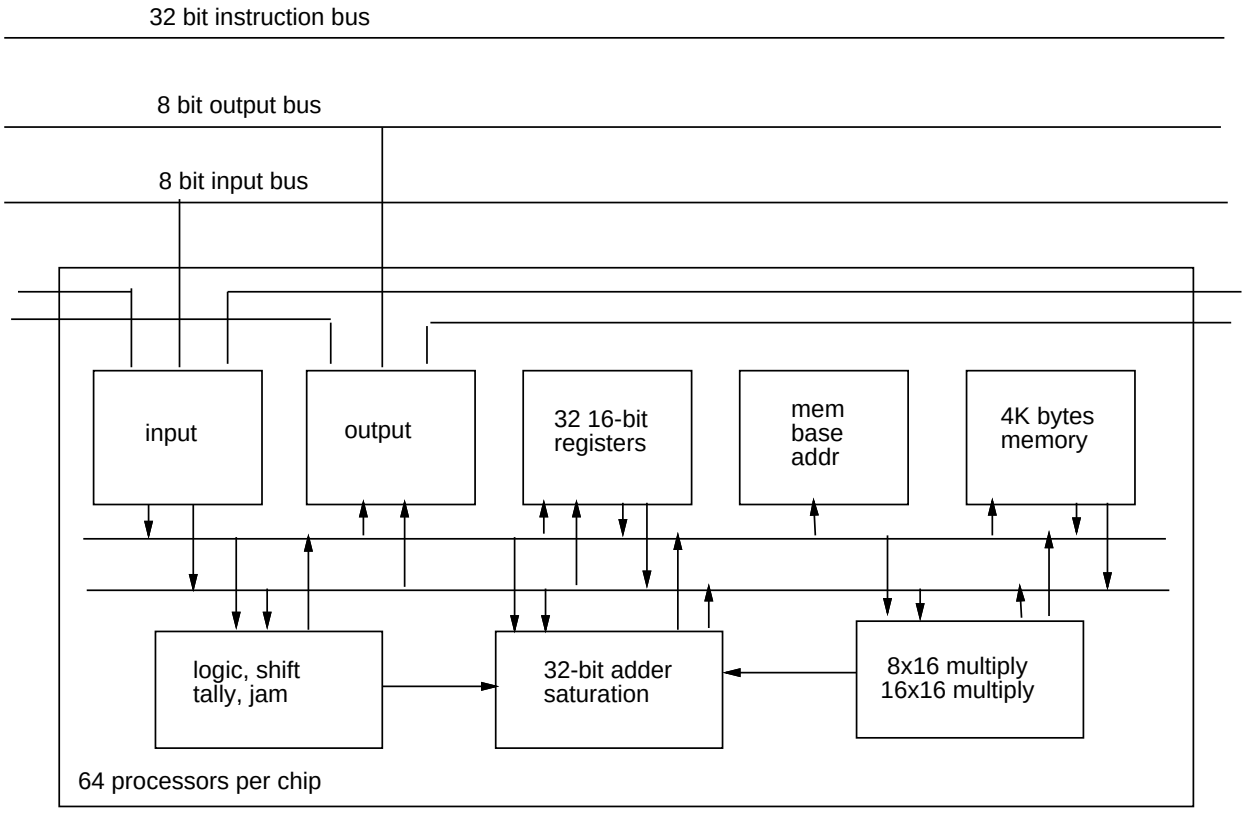
TInMANN (The integer markovian artificial neural network) [16] is a stochastic architecture proposed by the STONN group to implement competitive learning using stochastic computation. The system stochastically updates the weights with a probability proportional to the neural input, causing neurons closest to an input vector to move toward it and push others away. The algorithm replaces the proportional weight update of the standard Kohonen algorithm [44] with a probabilistic update that integrates to the same value over many cycles without requiring a multiplier.

The authors predict 15 MHz operation, and 145,000 two-dimensional training examples per second.

3.8 Adaptive Solution's X1 (CNAPS)

The Adaptive Solutions X1 chip (renamed CNAPS) [28] is a general purpose SIMD multiprocessor architecture [47, 46, 45] developed for neurocomputing applications. It achieves high performance (1.6 GCPS inner product, 1.28 GCPS and 300 MCUPS doing backpropagation, 12.8 GCPS using 1-bit weights) and can implement a wide variety of learning algorithms, as well as a wide variety of signal processing algorithms. A single chip has a fairly large synaptic capacity, storing 2 million 1-bit, 256K 8-bit, or 128K 16-bit weights equally distributed among 64 processors. Multiple chips can be assembled in various topologies.

The chip (see Fig 1) contains 64 processors, a 32 bit instruction bus, an 8 bit global output bus, an 8 bit global input bus, and a 4 bit inter-processor bus. Each processor includes 4k bytes of weight memory implemented using four-transistor SRAM, a memory base address unit, 32 16-bit registers, an input unit, an output unit, an 8x16 bit multipier, a 32 bit saturating adder, and a logic and shift unit.



25 MHz
64 processors/chip
1.6e9 CPS
1.2e9 BP CPS
0.3e9 BP CUPS
die size: several cm^2

0.8u CMOS
total memory 256K bytes
power < 4 watts/chip
scalable to many chips
learning on chip
saturation arithmetic
bit jamming, virtual zero memory

Figure 1: Adaptive Solutions X1 (CNAPS) architecture

The chip is quite large - over 5 cm^2. Redundancy is built in to improve yield. Each chip has 80 processors, only 64 of which need to work. The X1 is implemented in 0.8 micron CMOS. It is expected to run at 25 MHz, and dissipate about 4 watts, for an energy per connection of about 2 nJ.

The chip requires an external instruction sequencer. In typical configurations, a single sequencer will control a number of chips.

The X1 should turn out to be a useful research tool in studying the behavior of learning algorithms in larger networks.

3.9 Waferscale implementations

Raffel et al from MIT Lincoln Labs [63] discuss a possible waferscale network using multiplying digital to analog converters (MDACs) using their Restructurable VLSI (RVLSI) approach to waferscale integration.

Yasunaga et al of Hitachi [80, 81] present a completely digital implementation which uses reduced precision parallel arithmetic. It is implemented using a 0.8 micron CMOS waferscale gate array. It has 540 neurons and 64 synapses per neuron. The wafer has 9 bit neural activations and 8 bit synaptic weights. Each weight w_{ij} is stored as a pair (j, w_{ij}) to efficiently represent sparse networks.

The paper reports a "step speed" of 464nsec, which appears to indicate the system clocks at 2.15 MHz. A unique j is generated on each step; only those neurons with matching (j, w_{ij}) pairs participate in the computation. If all the weights are utilized, 64×576 connections can be processed in 576 "steps", for a performance of 64 connections per step, or 138 MCPS.

3.10 Neurocomputer implementations

There have been several papers describing implementations of neural nets on massively parallel processors or neural coprocessors. We only mention those that have reported performance by way of comparison with neurochips.

Blelloch and Rosenberg [12] discuss mapping backpropagation onto the Connection Machine, reporting 3 MCUPS. Pomerleau et al [61] discuss mapping backpropagation onto the CMU Warp, reporting 17 MCUPS. Watanabe et al [75] discuss backpropagation on the NTT AAP-2, reporting 18 MCUPS.

De Groot and Parker [27] describe mapping backpropagation onto the Lawrence Livermore SPRINT. The machine has 10^6 synapses, 64 transputers, dissipates 100 watts, and achieves 12 MCPS. That's 8μJ/connection.

3.11 Neural processor capacity and performance summary

who	ref	type	tech	capacity (N, S)	speed CPS	power chip, syn	learning	scalability (multichip)
Caltech	[2]	CCD	2μ	256, 64K	5e8	-	no	no
Chiang	[15]	CCD	3μ	144, 2016	1.44e9	2W, 1.4nJ	no	no
USC	[48]	cap	2μ CMOS	25, 2525	-	-	no	no
Matsushita	[56]	analog	2.2μ BiCMOS	64, 768	7.7e7	-	no	no
ETANN	[38]	analog	1.0μ EEPROM	64, 8192	8e9	2W, 250 pJ	no	no
NET32K	[24]	D,A	0.9μ CMOS	1024, 32K	3e11	160mW, 500fJ	no	yes
CLC	[5]	D,A	1.2μ CMOS	32, 1000	1e9	1W,1nJ	yes	yes
Mitsubishi	[9]	analog	1.0u CMOS	125, 10K	1e11	1.5W, 15pJ	yes	yes
JPL	[19]	analog	2.0μ CMOS	32, 1000	1e9	-	no	no
X1	[28]	digital	0.8μ CMOS	64, 256K	1e9	4W, 4nJ	yes	yes
Ouali	[59]	digital	2.0μ CMOS	-	-	-	no	yes
Duranton	[18]	digital	1.6μ CMOS	-	8e8	-	yes	yes
Hirai	[36]	digital	1.2μ CMOS	6, 84	8.4e4	-	no	yes
DNNA	[42]	stochastic	-	32, 1024	2e8	-	no	yes
STONN	[78]	stochastic	-	20, 2000	5e8	-	no	yes
TInMANN	[16]	stochastic	2.0μ CMOS	1, ?	-	-	yes	yes
Hitachi	[81]	digital wsi	0.8μ CMOS	540, 34K	1.4e8	-	no	yes
CM1	[12]	mpp	-	-	3e6	-	yes	-
Warp	[61]	systolic	-	-	17e6	-	yes	-
AAP-2	[75]	array	-	-	18e6	-	yes	-
SPRINT	[27]	systolic	-	-, 1e6	12e6	100W, 8μJ	yes	-
neuron	-	-	-	1e11,1e15	1e16	1W, 0.1fJ	yes	yes

Table 1: Summary of existing implementations

Table 1 gives a summary of existing implementations. It is not meant to be exhaustive; it is meant to be a sampling of a wide variety of implementations. Analog, waferscale, neurocomputer, and biological entries are for comparison purposes. The two numbers in the *power* column are chip power dissipation and energy per connection.

The remainder of this chapter discusses issues related to building digital VLSI neural networks. Large networks can easily exceed the capabilities of the technology, so it is important to try to maximize network performance and capacity while minimizing power consumption. One of the messages we will try to convey is that digital systems have certain strengths which can be exploited as technology scales down into the submicron regime.

4 Architectural weapons

The performance of digital VLSI systems can be enhanced using a variety of techniques. These are: pipelining, precision, iteration, concurrency, regularity, and locality. Of these, precision may be particularly useful in neural net applications, and iteration may become more widespread as technology scales down.

4.1 Pipelining

Neural networks are well suited to deep pipelining because the latency of an individual unit is not nearly as important as the computation and communication throughput of the system.

If a system is too heavily piped, too much area is taken in latches, and the load on the clock is too large, increasing clock skew and reducing the net performance gain.

A system clock based on the propagation delay of a 4:2 adder provides a good balance between area and perforamnce. This is typically 1/4 of a RISC processor clock period in the same technology.

4.2 Precision

Precision can have a significant impact on area, power, and performance. The area of a multiplier is proportional to the square of the number of bits. A 32×32 bit parallel multiplier is 16 times the area of an 8×8 bit multiplier. At 8 or fewer bits, a multiplier is just an adder. A Booth encoder can cut the number of partial products in half and a 4:2 adder can accept up to 4 inputs.

Precision should be leveraged where it is available in the system. For example, an $N \times N$ bit multiply generates a $2N$ bit result. A Hebbian style learning algorithm could compute a weight adjustment to higher precision than is available and choose a weight value probabilistically. Using "probabilistic update", our Boltzmann machine can learn with as little as 2 bits of precision in the weight store.

4.3 Iteration

Iteration involves using an area-efficient arithmetic element and a local high speed clock to compute a single result in several clock cycles. Iteration reduces logic area by time multiplexing resources at a higher frequency than can be managed globally.

The conventional way to achieve high computational throughput is to implement a parallel arithmetic element clocked at the system clock rate. An alternate approach is to generate a local clock which is at least some multiple of the system clock rate, and then to implement only a fraction of the arithmetic element.

The basic idea behind iteration can be illustrated by the following example. Suppose we want to build a multiplier which can accept two inputs and output a product on every clock cycle. Using conventional techniques, we would have to use N^2 full adders and clock the data through the multiplier at the system clock rate. Suppose, however, that we could generate a local clock which was at least twice the frequency of the system clock. We could then implement the multiplier in half the area using $N^2/2$ fulladders. Iterative structures trade space for time.

Iteration will become more widespread as technology scales down and local clock rates scale

up.

Mark Santoro implemented a 64X64-bit multiplier using an iterative 16X16-bit array [66, 67, 68]. He built the clock driver out of a 4:2 adder so it would be sure to track temperature and process variations. He achieved 400 MHz operation in 0.8μ CMOS.

Iterative structures, although highly area efficient, are likely to be higher power than unrolled structures. If unrolled, nodes only serve one function. Iteration essentially time multiplexes nodes. If the roles of the two nodes are unrelated, then their states are uncorrelated, and they may have a higher probability of changing state from iteration to iteration, dumping extra charge and thereby dissipating more power.

4.4 Concurrency

Concurrency is a widely used technique for increasing performance through parallelism. The degree of parallelism will increasingly be limited more by power density than by area. Neural networks are intrinsically highly parallel systems.

4.5 Regularity

Regularity in an architecture or algorithm permits a greater level of system complexity to be expressed with less design effort. Neural nets are composed of large numbers of similar elements, which should translate directly to VLSI implementation.

4.6 Locality

Local connections are cheaper than global ones. The energy required to transport information in CMOS VLSI is proportional to the distance the information travels. The available communication bandwidth is inversely proportional to wirelength, since the total available wirelength per unit area in a given technology is constant.

The energy required to switch a node is CV^2, where C is the capacitance of the node and V is the change in voltage. C is proportional to the area of the node, which for fixed width wires, is proportional to the length of a node.

5 Building blocks

A number of basic circuits and circuit techniques can be used to advantage in designing digital neural networks. This section is not intended to be a thorough presentation of digital logic design, as there are many excellent sources for this [54, 74, 77, 33]. Rather, we assume a familiarity with the basics and seek to highlight specific structures which are especially useful in designing digital neural network arithmetic elements, memory, and noise sources.

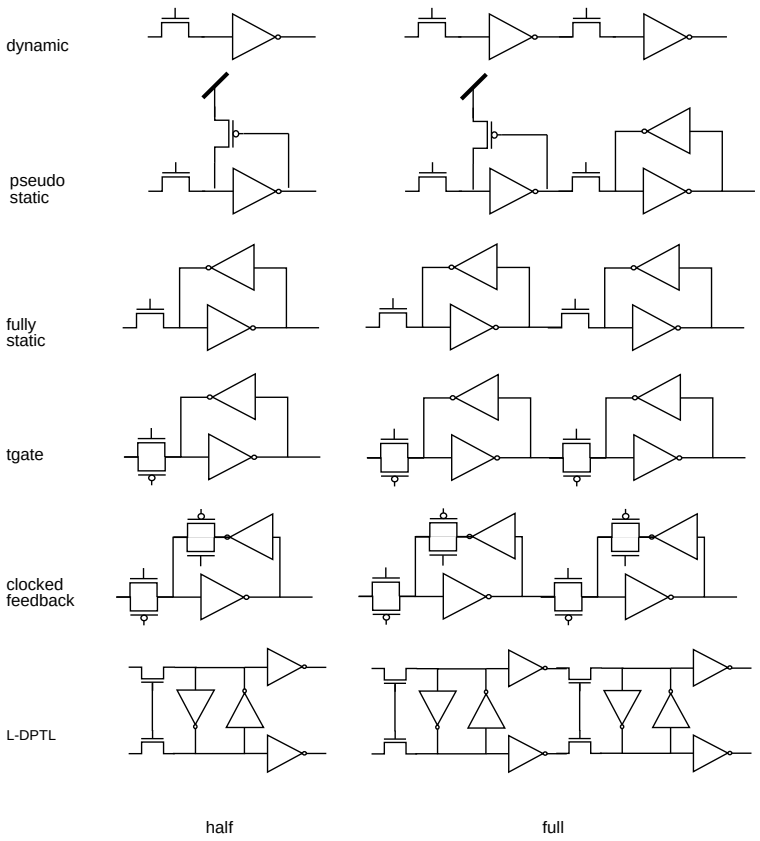


Figure 2: latches

5.1 Logic design styles

There are many logic design styles to choose from in implementing CMOS circuits. Logic design styles achieve different tradeoffs in speed, power, and area. The highest speed logic families also tend to consume the most power. The most compact tend to be slow.

Complementary pass-transistor logic (CPL) seems to offer modest performance, is compact, and low power. Yano et al's multiplier paper [79] has a good explanation of the style as well as detailed schematics of CPL arithmetic elements.

5.2 Latches and clocking

Latches play an important role in any digital design, especially if the design is pipelined. The right latch to use depends on the clocking discipline and the desired performance. Some clocking styles are safer than others. We have been using nonoverlapping two-phase clocks in our chips to date, but are using single phase clocking on our Boltmann chip.

Most of our designs use either the fully static or pseudo-static latch shown in Figure 2. There is a good section on latches in Weste and Eshraghian's book [77].

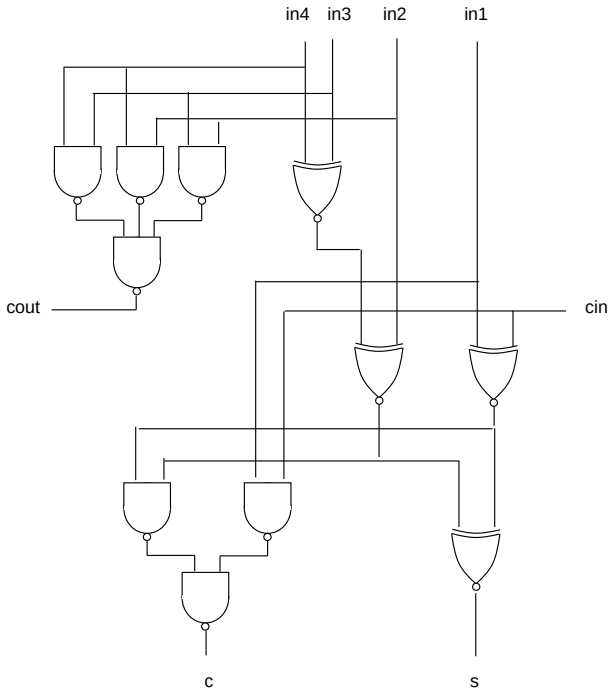


Figure 3: 4:2 adder

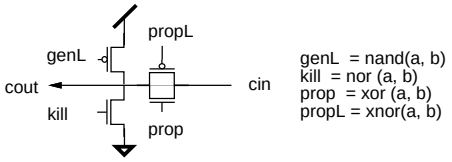


Figure 4: unloaded manchester carry chain

5.3 4:2 arithmetic

Carry propagation is an expensive operation in digital arithmetic. Several families of arithmetic have been developed to reduce the impact of carry propagation. Signed digit [10] and various redundant binary methods [29] have been proposed. We like "4:2" arithmetic based on 4:2 adders [70] because it interfaces cleanly to standard two's complement, and implements an efficient, compact accumulator [51, 50, 49, 66, 67, 68]. Although 4:2 adders can be implemented using two full adders, we discovered a "direct logic" implementation [51] (see Figure 3) that reduces the number of xors in series from four to three, increasing the speed by 33%.

5.4 Fast comparators

Unloaded manchester carry chains make very fast, efficient comparators. They are generally much faster than adders because there are no sum terms to load down the carry chain (see Figure 4). Comparators are needed in winer-take-all networks.

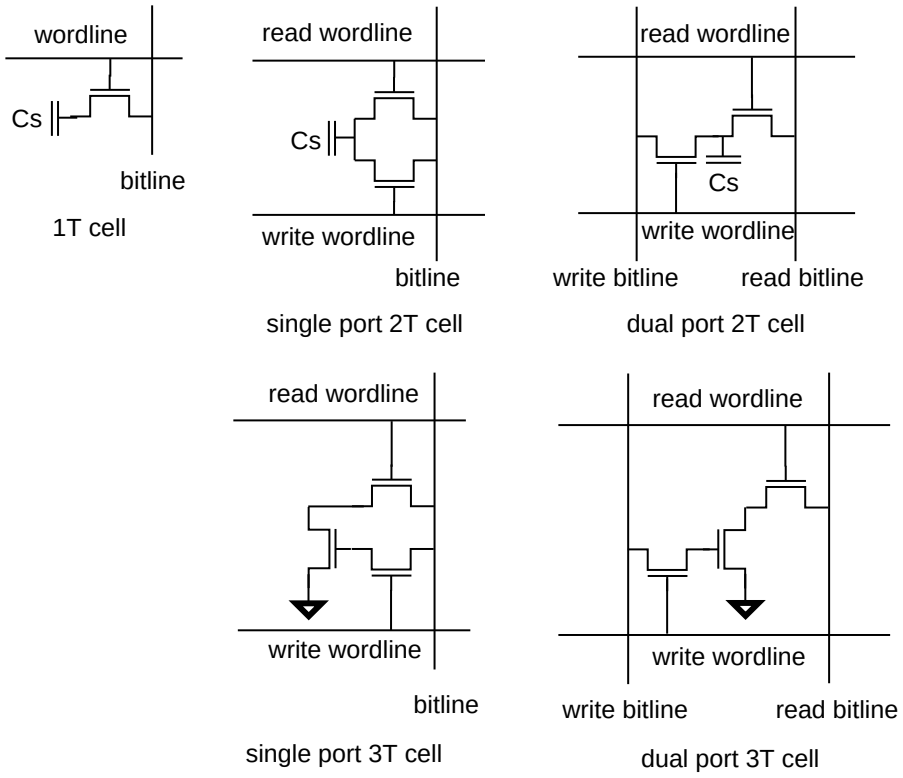


Figure 5: dynamic memory cells

5.5 Memory

Synaptic storage density is an important issue in large scale networks, so memory optimization is important. One-transistor (1T) dynamic random access memories (DRAMs) have the highest density. Six-transistor (6T) static memories (SRAMs) consume the least power. A typical 1T DRAM cell measures $6\lambda \times 11\lambda$ ($66\lambda^2$) [43]. A typical 4T SRAM measures $12\lambda \times 20\lambda$ ($240\lambda^2$) [3]. Shift registers can often be implemented with either SRAM or DRAM, saving substantial amounts of area and power.

5.5.1 DRAM

DRAM cells must be refreshed due to leakage current [14], and therefore consume more power than SRAMs. Normally the refresh power is a small fraction of the operating power, but could be significant in very large networks (see Section 10.

Figure 5 shows a variety of DRAM circuits. One-transistor (1T) DRAMs are the most compact but the most difficult to sense and control.

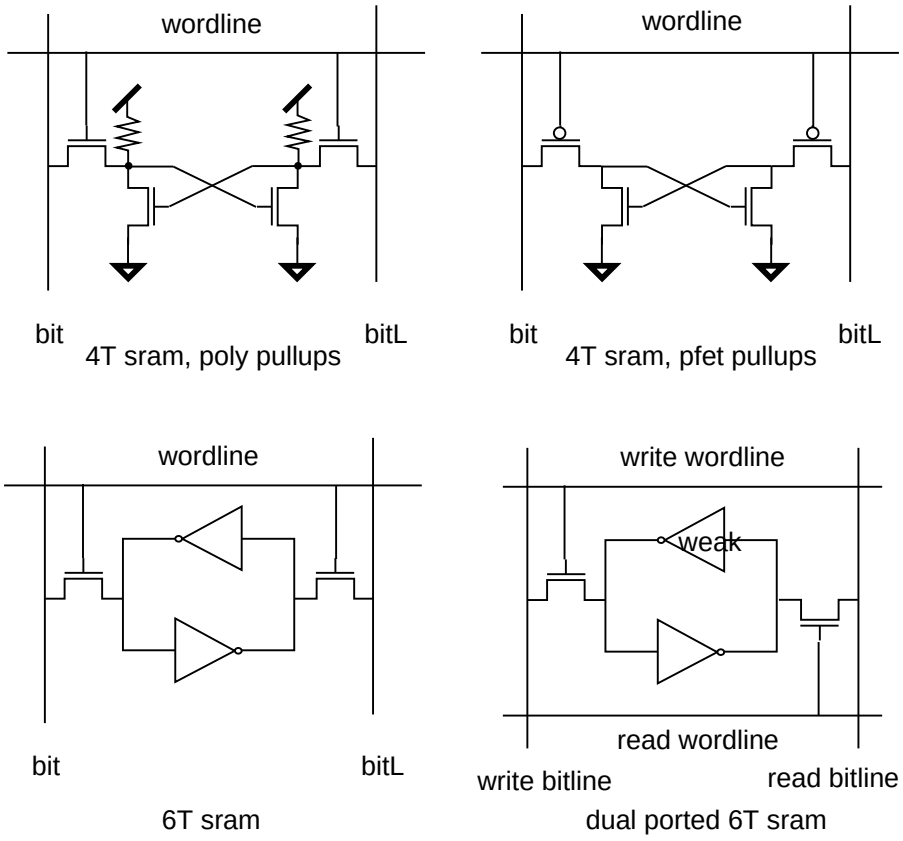


Figure 6: static memory cells

5.5.2 SRAM

Figure 6 shows a variety of SRAM circuits. Commercial SRAMs are normally implemented using a high resistance poly pullup, and can achieve densities only a factor of 4 worse than 1T DRAM.

5.5.3 Logic processes

The trench capacitors in DRAMs and high resistance poly pullups in SRAMs are not normally available in standard logic processes. A one-transistor DRAM was fabricated through MOSIS [23, 73], although the cell was quite large ($18.75\lambda \times 13.75\lambda$ ($258\lambda^2$).

5.5.4 Multi-level storage

Several researchers have reported techniques to store multiple levels in a single DRAM cell. The first were Heald and Hodges [31] in 1976. More recently, Aoki, Horiguchi, and colleagues [8, 40] reported 16 levels per cell in 1987. More levels might be achievable in neural networks since errors will always be small. Weight decay could be implemented by under-refreshing the memory.

5.5.5 Sense amplifiers

Figure 7 shows a high performance differential dense amplifier reported in [17], and recommended for use with 1T DRAMs. This sense amp has the nice property that the crosscoupled inverters automatically perform a refresh write after read.

Figure 8 shows a current mirror sense amplifier similar to one reported in Aizaki et al [3] in 1990.

Figure 9 shows a single ended sense amp which can be used with ROMs, 3T DRAMs, and single-ended SRAMs.

5.6 Noise sources

Noise plays an important role in many neural network algorithms. Generating a large number of uncorrelated noise sources is a hard problem. Joshua Alspector and colleagues have developed an area-efficient technique which they have implemented on their CLS chip [5]. They give this subject a thorough treatment in [6].

5.7 Stochastic computation

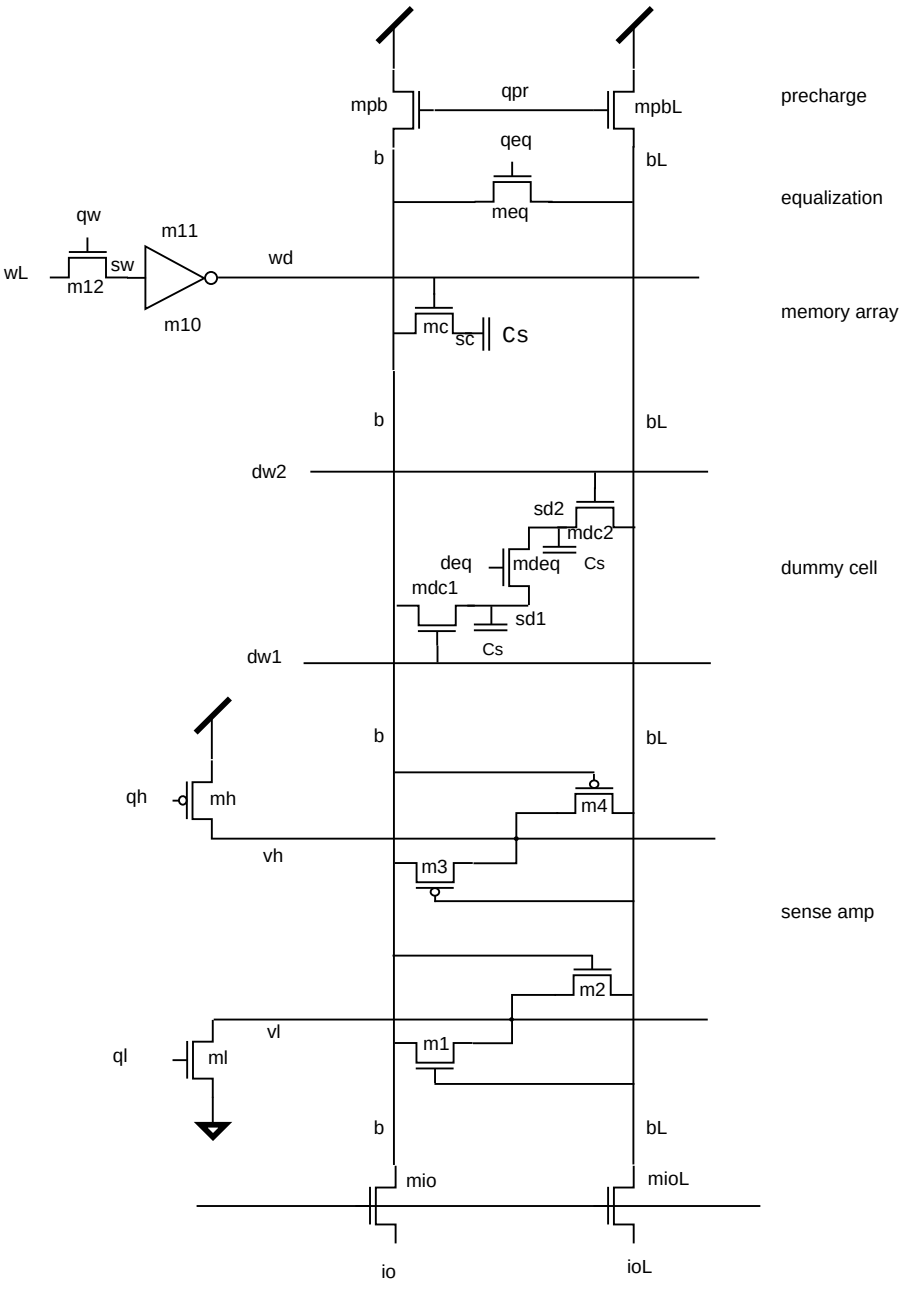


Figure 7: differential sense amplifier

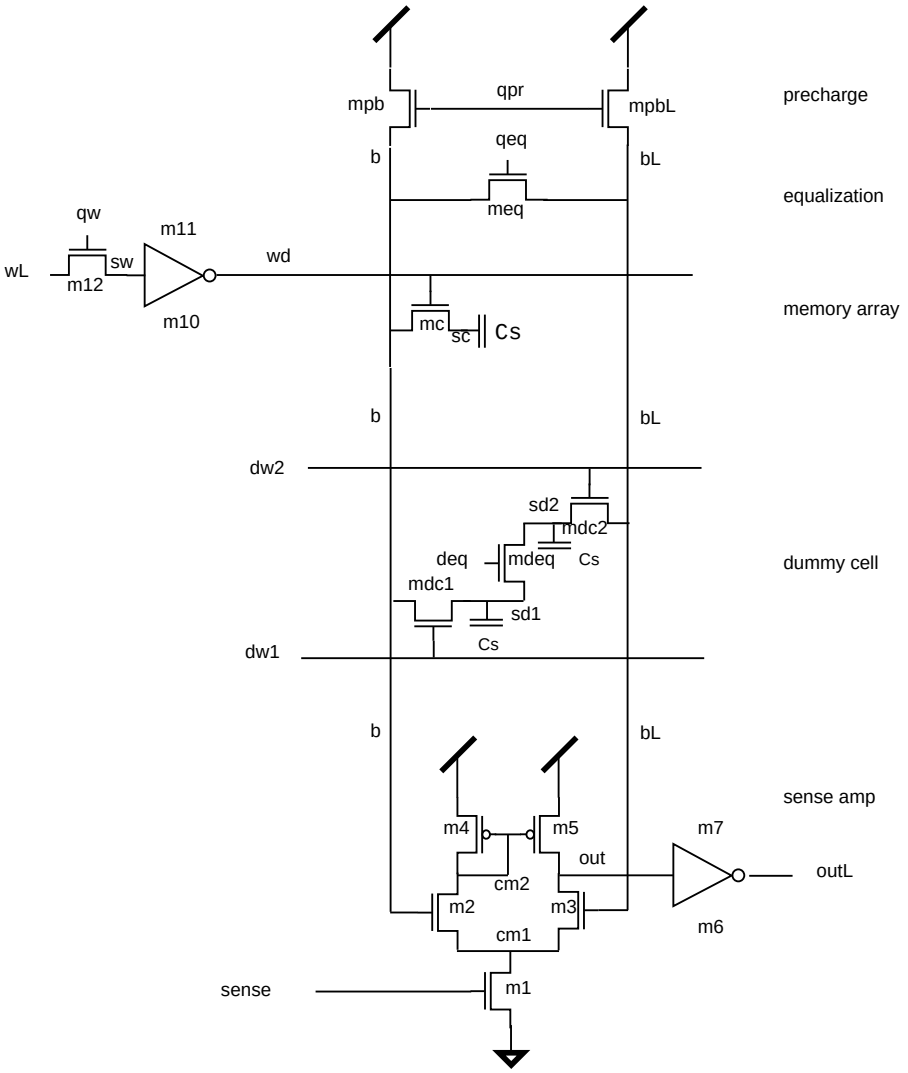


Figure 8: current mirror sense amplifier

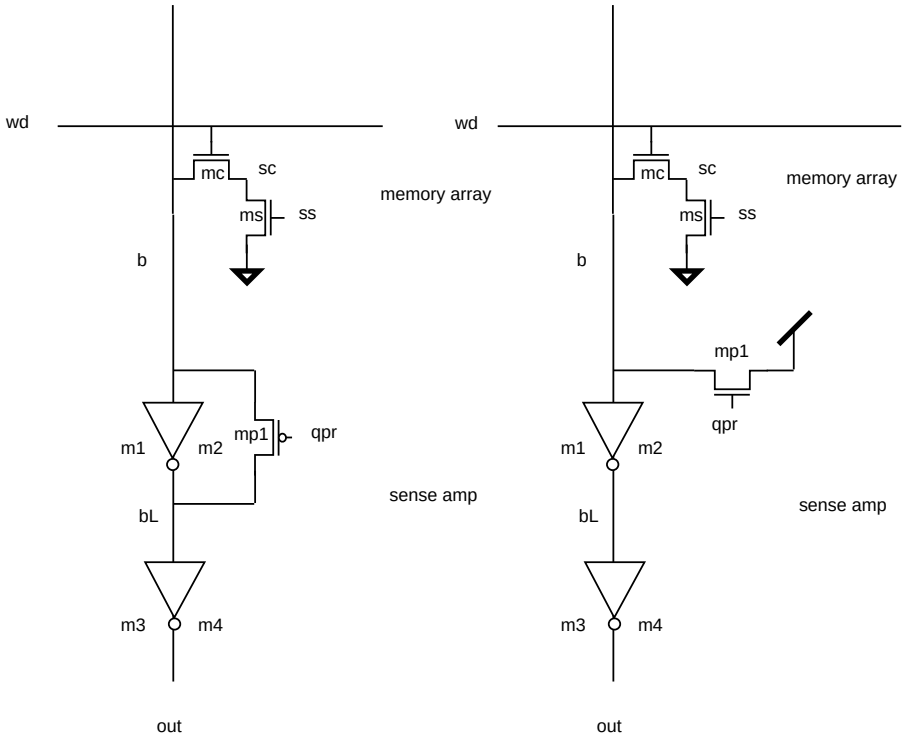


Figure 9: single ended sense amplifier

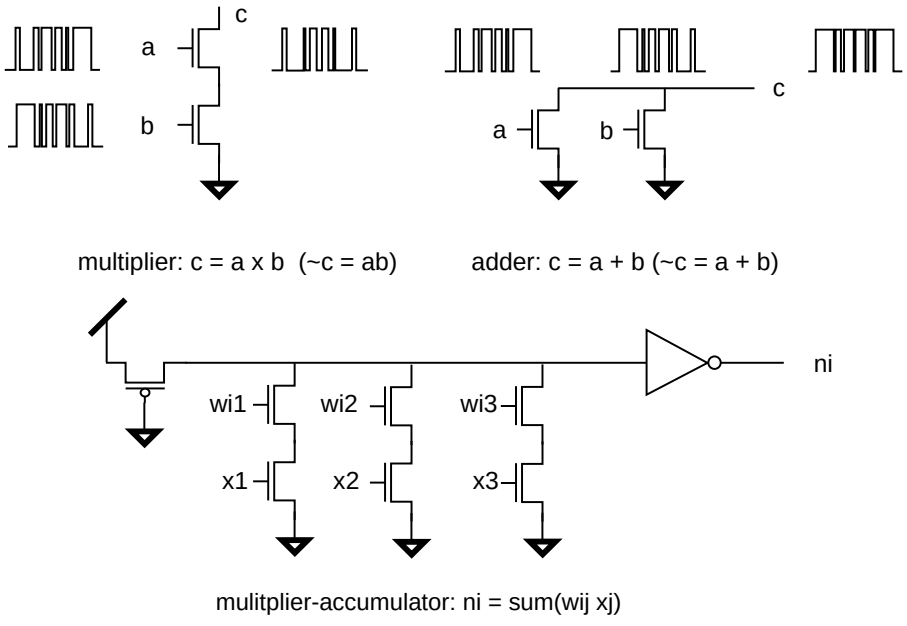


Figure 10: stochastic computing elements

Stochastic computation implements multiplication and addition using probabilities. It is particularly effective when a computation can be spread out over many cycles. This is often the case in neural network learning, where the network evolves incrementally over many cycles. Stochastic techniques have been applied successfully to Hopfield networks [42], competitive learning networks [16], and Boltzmann machines [7, 5].

Figure 10 shows the two basic operations: multiplication by ANDing pulse streams ($P(AB) = P(A)P(B)$), and addition by ORing ($P(A+B) = P(A) + P(B) - P(AB)$). The $P(AB)$ term in the probabilistic addition limits the available dynamic range of the computation.

Although stochastic techniques are very efficient at implementing local computation, they are much less efficient communicating globally because the energy required to transmit data with a dynamic range of N is proportional to N rather than to $\log(N)$ as in standard digital encoding.

Good discussions of stochastic computation can be found in [20, 52, 21].

6 Area, performance, and power estimation

We have developed a simple area, performance, and power estimation technique which we use to construct spreadsheets in the early stages of architectural exploration, feasibility analysis, and optimization of a new design. Area is computed by estimating the number of transistors required. Performance is estimated by building an RC timing model of the critical paths into the spreadsheet. Power is estimated using CV^2f, where f is obtained from the performance section of the spreadsheet.

Area, performance, and power are parametrized by technology. We have a "technology section" of the spreadsheet where we build in technology scaling rules to compute transistor transconductance and device parasitics.

6.1 Area estimation

We use a simple technique to estimate area of chips before we build them. We identify the major resources on the chip, and estimate the number of transistors for each resource. We then multiply the number of transistors in each case by an area-per-transistor which depends on how regular and compact we think we can make the layout. One-transistor DRAM and ROM are about $100\lambda^2$/xstr. 3T DRAM and 6T SRAM are about $200\lambda^2$/xstr. Tightly packed, carefully handcrafted logic is also about $200\lambda^2$/xstr. Loosely packed full custom logic is about $300\lambda^2$/xstr. Standard cells are about $1000\lambda^2$/xstr.

Block routing takes about 30% of the chip area. Standard cell routing takes 60% of the block. The pad frame reduces the die by about 1mm. For example, the largest die available on a standard MOSIS run is 7.9x9.2mm. Of this, 6.9x8.2mm, or 56.58mm^2, is available for logic and routing. Of this, 17mm^2 is routing, and 40mm^2 is logic. In 2 micron CMOS, lambda = 1 micron, so there is room for 133,000 transistors at $300\lambda^2$/xstr

The number of transistors required to implement a function can vary significantly depending on the design style. For example, a full adder implemented with gate logic requires about 30 transistors. However, it can be implemented in 15 transistors using pass-transistor logic. Which is best depends on desired performance and power dissipation, input drive and output load.

We maintain a list of leafcells, the number of transistors they require, and their area-per-transistor, which we reference in estimating requirements of new designs.

We also have a set of "tiling functions" which we use to construct complex blocks. For example, an $N \times M$ bit multiplier requires roughly NM full adders, whether it is implemented as an array or a tree.

This technique is especially well suited to spreadsheet implementation, and is especially useful during the early stages of architectural exploration and feasibility analysis in an area-limited design.

6.2 Performance estimation

We estimate performance using a simple RC timing model based on the RSIM simulator [72], in which transistors are calibrated to have an effective resistance charging or discharging a node capacitance.

We build the following equations into our spreadsheet models to allow the performance estimates to scale with technology. The symbology of these equations follows the development in Hodges and Jackson [37].

```
Effective resistance of transistors

    Ilin = k/2(2 * (Vgs-Vt) * Vds - Vds^2)   Vds <= Vgs - Vt
    Isat = k/2(Vgs - Vt)^2                   Vds >= Vgs - Vt

    Iav  = integrate(I * dt)/T
    reff = DV / Iav
         = const * k

    kn   = un * cox          kp   = up * cox          A/V^2
    rn   = 1/kn/(Vdd - Vt)   rp   = 1/kp/(Vdd - Vt) ohms/sq
    Rn   = rn * l / w        Rp   = rp * l / w       ohms

Parasitic capacitance

    cg   = eox / tox                                 farad/m^2
    cox  = cg                                        farad/m^2
    xj   = (2*esi/q/NA*(V-Vt))^.5                    meters
```

```
    cj   = esi / xj                                  farad/m^2
    cjsw = 3 * cj * xj = 3 * esi                     farad/m
    ci   = eox / hi                                  farad/m^2
    cisw = ci * ti                                   farad/m

Propagation delay

    tpu  = Rp * (Cd + Ci + Cg)                       sec
    tpd  = Rn * (Cd + Ci + Cg)                       sec
```

6.3 Power estimation

There are three principal components to power dissipation in most CMOS systems:

$$P_{dc} = V^2/R_{dc}$$

$$P_{sc} = I_{sc}V$$

$$P_{ac} = CV^2f$$

where

V is the supply voltage
C is the total capacitance being switched
f is the clock frequency
R_{dc} is the total static pullup or pulldown resistance
I_{sc} is the short-circuit current
P_{dc} is the power dissipated at DC
P_{sc} is the power dissipated due to short-circuit current
P_{ac} is the power dissipated by switching capacitance.

P_{dc} can be designed out of a system, except leakage, which is usually on the order of a few microwatts [14, 30], but P_{sc} and P_{ac} cannot. In a CMOS inverter, I_sc is the current which flows when both the nfet and pfet are on during switching. Powell and Chau [62] have reported that P_{sc} can account for up to half the total power.

We have done some investigations which suggest that the short circuit current can be significant if rise times are long and transistors large, and that in most cases P_{sc} can be reduced to less than 10% of total power by sizing transistors. This implies that short circuit current can be more of a problem in gate array or standard cell design, where transistors are fixed sizes or are sized to drive large loads.

Consider a single CMOS inverter driving a purely capacitive load. Initially, assume the gate is at 0 volts, so the pfet is on, the nfet is off, and the output is at 5 volts. Now, switch the gate from 0 to 5 volts. Ideally, the pfet should turn off instantly, the nfet should turn on and

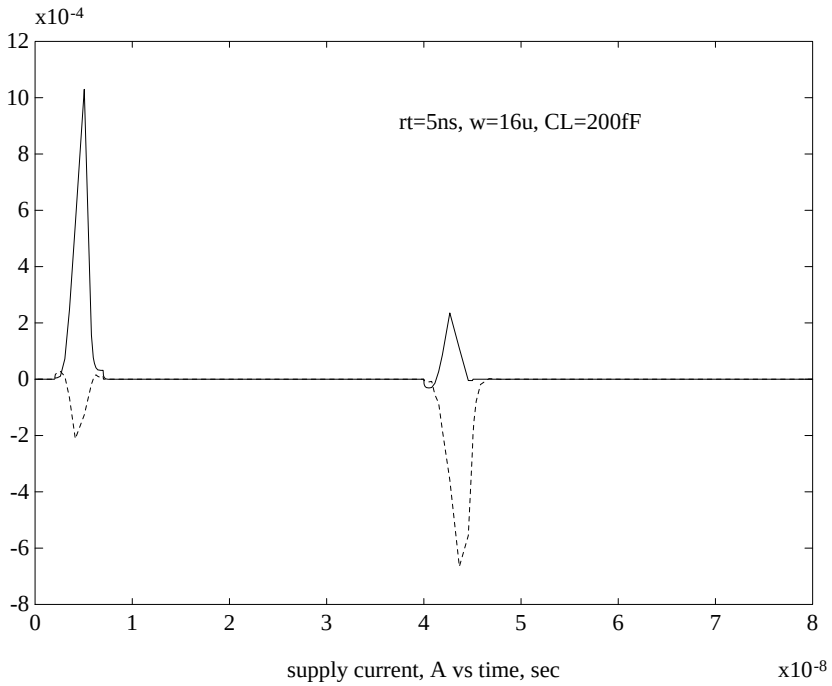


Figure 11: short circuit current

drain the charge off the output (actually supply electrons to the output) until the output potential reaches 0 volts. The work done (or energy consumed) by the inverter is just QV where Q is the charge on the output and V is the initial potential difference between the output and GND. But $Q = CV$ so the work done is CV^2.

In practice, the input does not switch instantly, so both the nfet and the pfet are on for a short time, causing excess current to flow.

We measured the short circuit charge for a variety of transistor sizes, rise times, and output loads using spice [58] on a typical 2 micron CMOS technology from MOSIS.

Figure 11 shows the current flowing through vdd and gnd supplies as the gate is switched first from 0 to 5 volts between 2ns and 7ns, and then from 5 to 0 volts between 40ns and 45ns. The short circuit current in each case is the smaller spike; it is the current flowing through the supply which should be off. The short circuit charge is the area under the short circuit current. In the figure the short circuit charge is about 15% of the charge initially on the output, so P_{sc} will be about 15% of P_{ac}.

Figure 12 shows short circuit charge as a percentage of output load vs input rise/fall time. Each graph has a pair of curves for each of 5 output loads: 0, 100, 200, 500, and 1000 fF. One curve in each pair is for a rising input, the other for a falling input. The short circuit charge for a given output load is nearly the same whether the input is rising or falling. The pair with no output load has the largest short circuit charge.

The short circuit charge is negative for fast rise times because the coupling capacitance between the gate and the fet drains pulls the output above 5 volts. This deposits additional

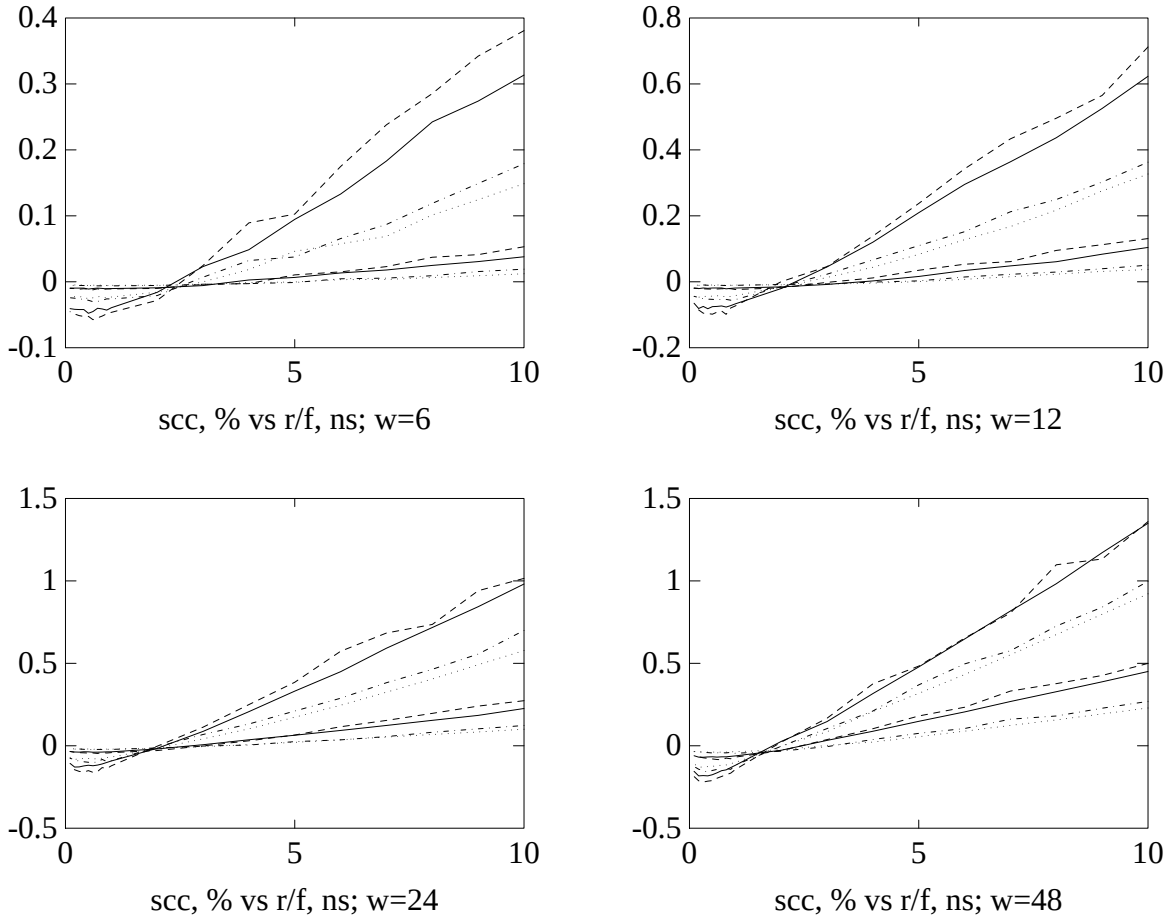


Figure 12: short circuit charge as a percentage of output load vs rise time for different size devices

charge on the output which must then be removed. So there is an energy cost associated with switching an input too fast, and a finite rise time at which the net short circuit charge is exactly zero.

We want to investigate this phenomenon in more detail, and see if we can modify our transistor sizer to take advantage of it. We also want to extend our analysis to complex gates, and other logic design styles.

In practice, we estimate power in a variety of ways. In some cases, when we have experience or know statistically what percentage of the nodes are switching, we use CV^2f. In other cases, we use Powell and Chau's power factor approximation (PFA) technique, which uses the energy of existing devices to predict new ones. In still other cases, when the devices have an analog behavior, such as sense amplifiers in memories and reduced voltage swing logic, we use spice to compute current and integrate to find charge dumped.

We have modified the RSIM timing simulator to accumulate the charge dumped as nodes switch during simulation. This is easy to do if the simulator is event driven. We compared RSIM's results on a signal processing chip we fabricated through MOSIS [13] with power measurements done on a performance tester and found agreement to within 20%.

6.4 Technology scaling rules and examples

To scale area, performance, and power to a desired technology, we build into our spreadsheet the parameters of a base technology (2.0 micron CMOS in our case; $\lambda_0 = 1.0\mu$), compute $S = \lambda_0/\lambda$, and apply the equations in table 2. We leave V, the supply voltage, explicit so we can compare the impact of "constant voltage" scaling and full scaling. These equations follow the development in Hodges and Jackson [37].

Table 3 shows how actual 2μ numbers would look scaled for a range of technologies. In the table, R(12:2) is the effective resistance of a $12\lambda \times 2\lambda$ transistor. Cg, Cd, and Ci are nominal gate, diffusion, and wiring capacitances. tgate is the propagation delay of a single gate. t4:2 is the propagation delay of a 4:2 adder.

Table 4 shows typical spice parameters for different technologies. The "exp" row in the table shows the exponent which would be supplied to spice. For example, for 2.0μ, tox=403e-10. cpa, $m1a$, and $m2a$ are area capacitance of polysilicon, metal1, and metal2 in farads/meter^2.

The information for 2.0, 1.6, and 1.2 μ technologies was obtained from MOSIS. 16 runs were averaged in each technology, and the deck closest to the mean was selected as nominal. The 0.8 micron information was obtained from a much smaller dataset.

These technologies all run at 5 volts, so S^2 performance is predicted by the formulas in Table 2. However, Table 4 shows that mobility, uo, is not constant. Comparing 2.0μ and 0.8μ, uo is scaling as $1/S^{1/2}$. Now R scales as $1/S/V/uo = 1/S^{1/2}/V$ and C as $1/S$ so propagation delay tp should then scale as $1/S^{3/2}/V$.

According to Hodges and Jackson, mobility is a function of substrate doping, N_A. N_A increases as λ decreases. As N_A increases, collisions become more likely so mobility decreases.

param	scaling	description
tech	S	
tox	$1/S$	gate oxide
cox	S	gate capacitance
uo	1	mobility
k	S	transconductance
r	$1/S/V$	resistance
xj	$V^{1/2}$	junction depth
cj	$1/V^{1/2}$	diffusion capacitance
cjsw	1	diffusion sidewall capacitance
hi	$1/S$	metal elevation
ti	$1/S$	metal thickness
ci	S	interconnect area capacitance
cisw	1	interconnect sidewall capacitance
R	$1/S/V$	device resistance
C	$1/S$	device capacitance
tp	$1/S^2/V$	propagation delay
I	SV^2	current
i	S^3V^2	current density
p	SV^3	power per device
P	S^3V^3	chip power dissipation

Table 2: Technology scaling

tech		2.0	1.6	1.2	1.0	0.8	0.6	
S		1.0	1.3	1.7	2.0	2.5	3.3	
S^2		1.0	1.7	2.9	4.0	6.3	11.0	
clk	S^2	20	34	58	80	125	220	MHz
clk	S^2	80	136	232	320	500	880	MHz
R(12:2)	$1/S$	6.0	4.6	3.5	3.0	2.4	1.8	kohm
Cg	$1/S$	20.0	15.4	11.8	10.0	8.0	6.1	ff
Cd	$1/S^2$	40.0	23.5	13.8	10.0	6.3	3.6	ff
Ci	$1/S$	40.0	30.8	23.6	20.0	16.0	12.2	ff
tgate	$1/S^2$	1.38	0.81	0.48	0.35	0.23	0.13	nsec
t4:2	$1/S^2$	12.5	7.35	4.31	3.13	1.98	1.14	nsec

Table 3: scaling example

tech	u0n	u0p	cgdon	cgdop	cjswn	cjswp	xjn	xjp	nsubn	nsubp
exp	0	0	-12	-12	-12	-12	-9	-9	+15	+15
2.0	631	237	298	285	548	334	250	50	5.76	6.24
1.6	583	186	573	494	588	184				
1.2	574	181	628	324	423	159				
0.8	447	101	229	271	200	200	157	138	85.58	79.65

tech	tox	cpa	cjn	cjp	m1a	m2a		
exp	-10	-6	-6	-6	-6	-6	tox	$1/S$
							cpa	S
2.0	403	388	130	262	26	19	cjn	$S^{3/2}$
1.6	250	573	140	432	35	23	cjp	S
1.2	209	644	293	481	36	24	m1a	S
0.8	170	794	547	570	79	31	m2a	$S^{1/2}$

Table 4: nominal process parameters for different technologies

tech	tpd	MHz constV	Vdd	MHz scaleV
2.00	10.83n	92	5	
1.60	7.58n	132	5	
1.40	6.12n	163	5	
1.20	4.78n	209	5	
1.00	3.57n	280	5	
0.80	2.50n	400	5	
0.60	1.58n	634	3.3	418
0.50	1.18n	849	3.3	560
0.40	825p	1213	3.3	801
0.30	520p	1921	3	1153
0.25	389p	2572	3	1543
0.10	90p	11143	2	4457

Table 5: 4:2 adder-based clock circuit performance

If the increase in N_A is necessary to prevent breakdown in the presence of higher E fields in constant voltage scaling, then presumably NA could be kept constant if V were scaled.

Table 5 shows predicted performance of Mark Santoro's self-timed 4:2 adder based clock driver [68] in various technologies based on both constant V ($S^{3/2}$) and scaled V (S). The numbers are in good agreement with observed performance at 2.0, 1.6, and 0.8μ CMOS. Extrapolations below 0.5μ are highly speculative. They serve as an upper bound on expected performance.

Process variations can impact performance significantly. We have found the ratio of "fast-fast" models to "slow-slow" models to be a factor of two in performance, with "typical" models right in the middle.

$$ff/typ = typ/ss = \sqrt{2}$$

$$ff/ss = 2$$

Performance degrades by a factor of 1.8 from 25degC, 5V to 130degC, 4.5V. Worst case performance can therefore be as much as four times slower than best case when process and temperature variations are combined.

7 Design techniques for maximizing performance

	const V	scale V
devices/area	S^2	S^2
speed	S^2	S
ops/sec/area	S^4	S^3
energy/op	$1/S$	$1/S^3$
power/area	S^3	1

Table 6: Energy scaling

For many problems in signal processing, performance can be maximized by maximizing pipelining and parallelism. This is an excellent technique for small and medium scale systems with generous power budgets. But as systems get very large, the level of parallelism achievable either by pipelining or replication will be limited by power considerations. Table 6 suggests that scaled voltage is the way to go for implementing very large networks.

In highly pipelined systems, performance is maximized by placing pipe stages so delay is the same in every stage. This requires good timing analysis and optimization tools.

As technology scales, the clock rates achievable by highly pipelined structures will be difficult to distribute globally. Self-timed iterative circuits can save area by performing high speed local computation.

8 Technology scaling of VLSI neural networks

We have applied the area and performance estimation techniques described in Section 6 to three different inner product processor architectures. The first (SP; see Figure 13) has one processor per synapse. The second (NP; see Figure 14) has one processor per neuron, with the synaptic weights stored in memory local to each processor, similar to the Adaptive Solutions X1 architecture. The third (FP; see Figure 15) has a fixed number of processors on chip, and off-chip weights.

SP has the lowest synaptic storage density but the highest computational throughput. FP has the lowest throughput, but the highest synaptic storage density and unlimited capacity.

The number of processors which can be placed on a single FP is I/O limited. Assuming 4 bit weights, 128 pins would be required to support 32 processors. The I/O constraint can be substantially alleviated with multichip module (MCM) packaging [41]; this allows far more flexibility in choosing die size and system partitioning.

8.1 Technology and performance scaling of inner product processors

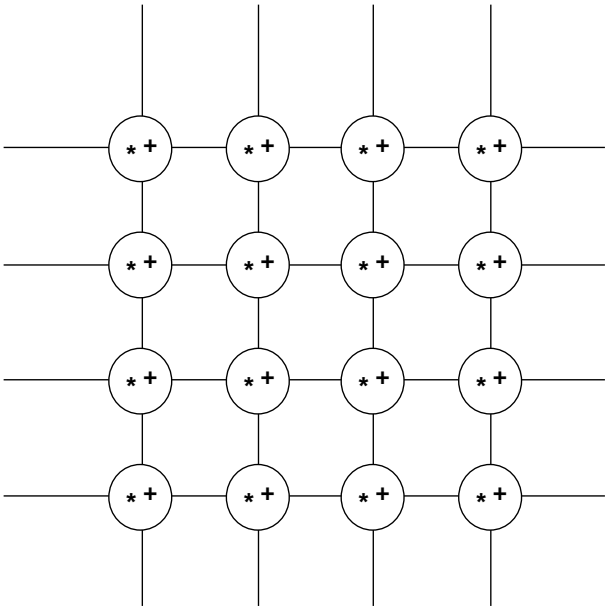


Figure 13: SP: one processor per synapse

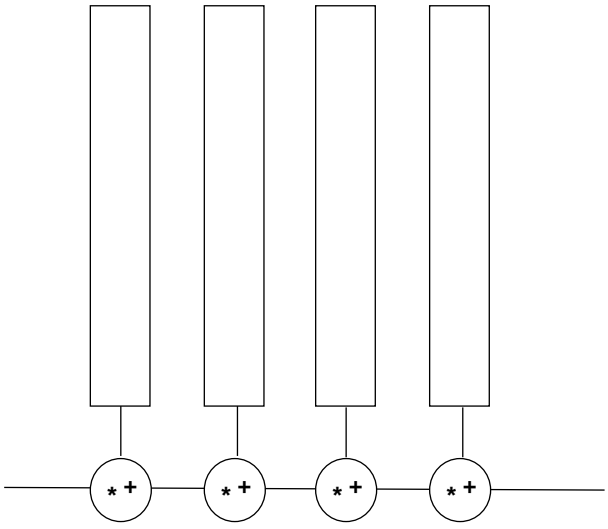


Figure 14: NP: one processor per neuron

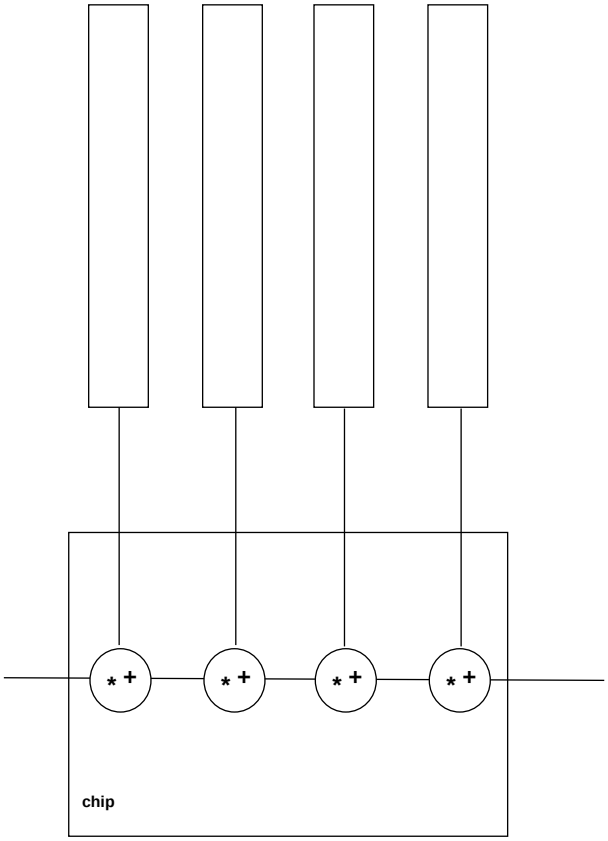


Figure 15: FP: off-chip weights

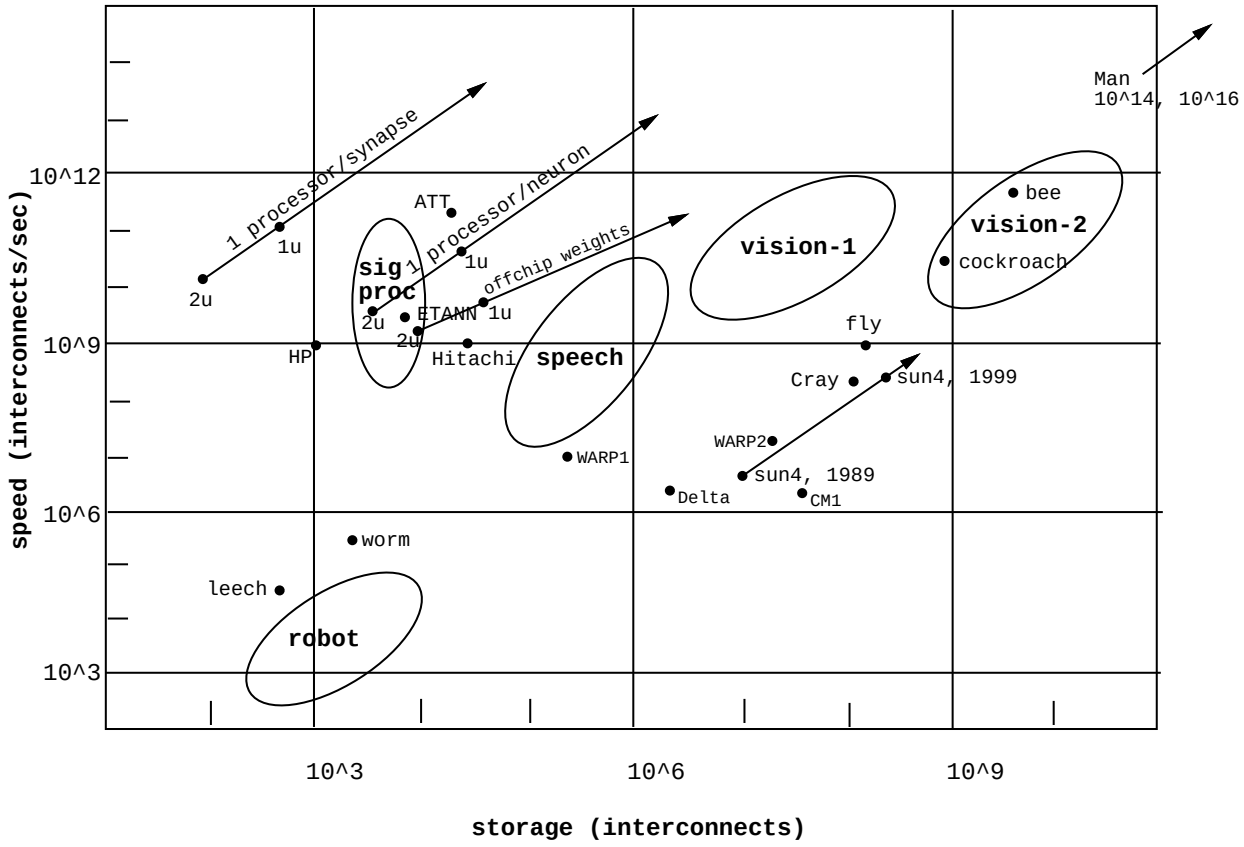


Figure 16: technology and performance scaling

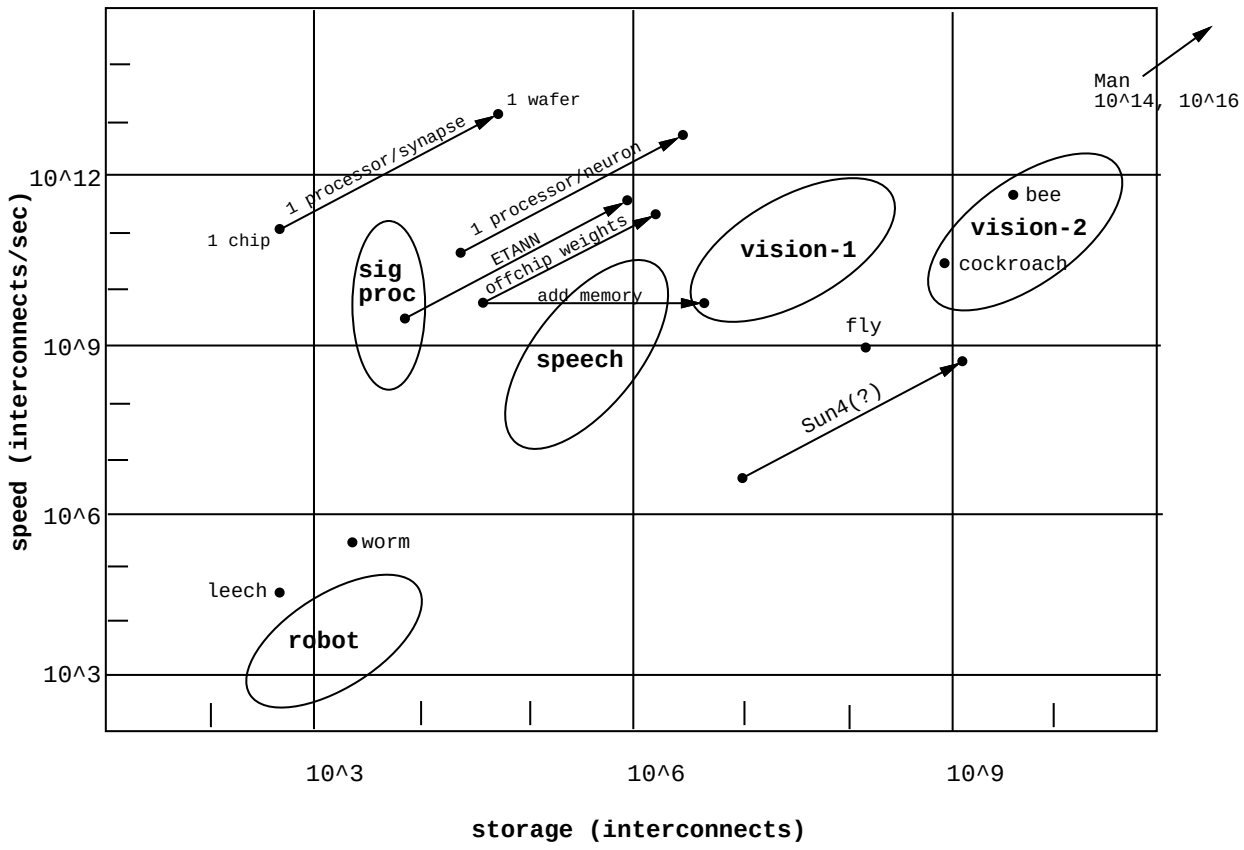


Figure 17: waferscale integration at 1.0μ

In Figure 16 we plotted the technology scaling trendlines of the SP, NP, and FP architectures on the DARPA-style capacity-vs-performance graph, along with the DARPA application requirements.

In Figure 17, we plotted the waferscale integration trendlines of the three architectures. The FP architecture has two trendlines. Since its weights are offchip, a waferscale implementation has the option of replicating memory alone. In this case performance remains constant, but capacity improves. Interestingly, of the options shown, this one most closely matches the requirements of the DARPA applications.

In Figure 18, we derive the architecture from the application and the technology. In a very rough sense, an application can be characterized by its capacity and performance requirements. An architecture can be loosely defined in terms of the connections serviced by a processor. For example, the SP architecture has one connection per processor. The technology defines the available performance. According to Figure 18, the product of the available performance and the ratio of the desired capacity and performance determines the architecture.

Figure 18 suggests that SP is most appropriate for the signal processing application in 2.0 micron CMOS, but that NP is more appropriate in 1.0 and 0.6 micron. By comparing chip DRAM capacities in the table (5 bit weights), we see that the FP architecture is best in all

Applications and architectures

connections/processor = connections/system / connections/sec/system X connections/sec/proc

(architecture) (application) (technology)

Application	connections	cps	connections / cps	cpp		
				2.0u	1.0u	0.6u
robot arm	3e3	3e4	1e-1	8e6	2e7	4e7
signal proc	3e4	1e10	3e-6	2.4	600	1500
speech	1e6	1e9	1e-3	8e4	2e5	5e5
vision-1	1e8	1e11	1e-3	8e4	2e5	5e5
vision-2	1e10	1e11	1e-1	8e6	2e7	5e7

Technology capacity and performance

		chip		wafer	
Tech	Clock(S^1.5)	3TDRAM	1TDRAM	3TDRAM	1TDRAM
2.0	80 MHz	1.5e3	4.5e3	1.5e5	4.5e5
1.0	200 MHz	6.0e4	1.8e5	6.0e6	1.8e7
0.6	500 MHz	2.0e5	6.0e5	2.0e7	6.0e7

Application	Nprocessors	Nchips (wafers)
robot arm	e-4	e-2 chips
signal proc	50	1 chip
speech	5	10 chips
vision-1	500	10 wafers
vision-2	500	1000 wafers

Figure 18: algorithms and architectures

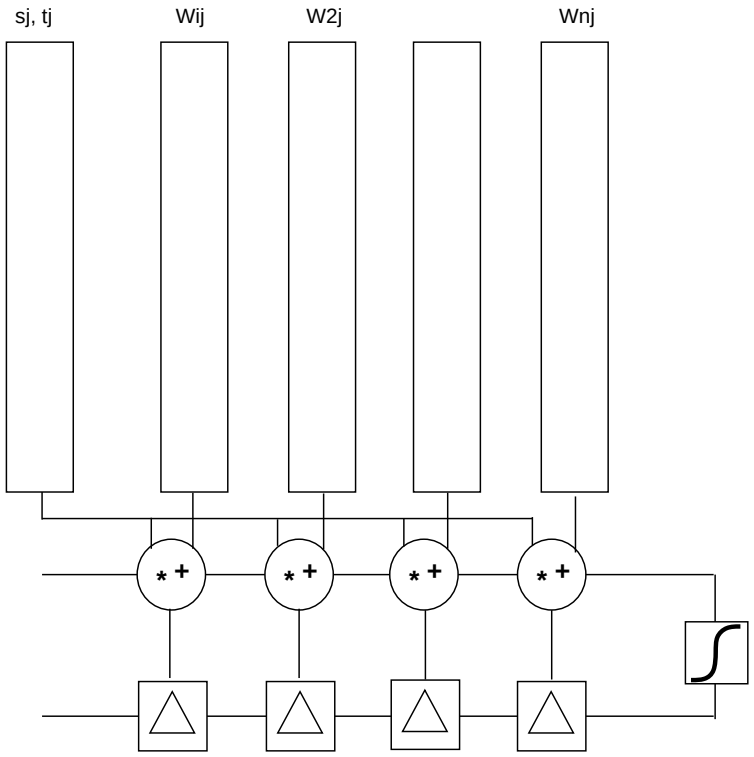


Figure 19: the Stanford Boltzmann Engine

other cases except speech and vision-1 in 0.6 micron, where NP is best.

9 The Stanford Boltzmann Engine

Boltzmann machines [34, 1, 7, 35, 60] are a special class of neural networks whose learning algorithm can be shown to minimize a global energy measure using only local information. The network contains recurrent connections (feedback), which can result in multiple responses to a single stimulus depending on the time evolution of the neural activations in the presence of noise.

Alspector has shown that both the deterministic (mean field) and stochastic variations of the Boltzmann learning algorithm can learn NETtalk [69] as well as the backpropagation algorithm. He says the computational effort of mean field learning is 3× backprop; stochastic is 100×.

Three Boltzmann chips have been implemented to date [7, 9, 5]. Each uses an analog current sum and adaptive synaptic processors.

We are implementing a fully digital Boltzmann engine (see Figure 19) which stores synaptic values and neural activations in one-transistor dynamic RAM and implements the $w_{ij}x_j$ products with 64 pipelined 5×5-bit multiplier accumulators operating in parallel. We are designing for 80 MHz operation, 100 pJ per connection in 2 micron CMOS, and a feedforward

computation rate of 5 GCPS.

During Boltzmann learning, the chip will complete one anneal step every 800 nsec. With 32 temperatures in the anneal schedule (we have found that an N neuron network needs about $N/2$ temperatures in the anneal schedule), the chip will process one training vector every 25 μsec, and learn to identify it in 100 to 1000 presentations. Because the implementation is purely digital, previously learned weights can be up- and down-loaded between the chip and a host.

Our simulations indicate 5 bits of precision are sufficient for both neural activations and synaptic weights, and that as few as 3 bits can be used with "probabilistic update", wherein an available weight value is selected with probability proportional to a higher precision accumulated inner product.

We plan to implement a single sigmoid unit as a lookup table. We plan to implement the temperature multiplier using reduced precision floating point with a 2 bit mantissa and a 4 bit exponent.

The multiplier-accumulators accumulate in carry-save form using 4:2 adders to reduce hardware and increase performance. To avoid delivering data to the sigmoid unit in carry-save form, a few extra cycles are appended to the accumulation to flush the carries.

There are two weight update processors, since N^2 weights need to be updated every $N^2/2$ cycles (N cycles per inner product product $\times N/2$ temperatures).

10 Large networks

VLSI faces formidable obstacles implementing large neural networks. Although the networks are massively parallel, there is evidence that communication is predominantly local [11, 64, 53]. Furthermore, neurons are slow compared to VLSI, assuming a simple neuron model, implying that computation and communication should be time multiplexed. Digital implementations lend themselves well to time multiplexing, but this requires a projection or folding of the network onto the hardware.

Rudnick and Hammerstrom of the Oregon Graduate Center Cognitive Architecture Project wrote an excellent paper on a waferscale architecture for large scale neural networks [64]. They describe a waferscale communication architecture involving dual concentrate/broadcast domain trees. They assume that most communication is local, and that when a neural activation changes, it is broadcast to the smallest domain that completely contains its fanout. Most often, this is just its nearest neighbor, but it could be the entire network.

10.1 Energy/power

Carver Mead wrote an excellent paper for the October 1990 Proceedings of the IEEE [53], in which he discusses the energy requirements of electrical and biological systems. He makes

the point that in systems on the scale of human cortex, with 10^{16} synapses, the energy budget is a primary constraint. The human brain consumes a few watts, so each synaptic computation is constrained to require on the order of 0.1fJ.

In 2μ CMOS, a single minimum size (4×2) transistor is 7fF. That translates to 175fJ at 5V. C_g scales as $1/S$. In 0.5μ CMOS, a single minimum size transistor is 1.75fF, or 16fJ at 3V. Projecting a few years down the road to 0.1μ technology, and 1V supplies, a single transistor might consume 0.350fJ.

Mead maintains that the energy required to switch a single transistor is comparable to the energy to perform one complex synaptic computation in a biological network, and that digital systems switch about 10000 transistors to perform a single operation, making them 10000 times less efficient than biological systems.

A 5x5 bit multiplier-accumulator can be built using roughly 1000 transistors such that only about 100 switch during a computation, so we might hope to get within a factor of 100 of biology using reduced precision digital arithmetic.

There is also an energy cost associated with retrieving synaptic weights from a memory structure. If the synaptic store is divided into 64x64 bit blocks, only one of which is accessed, about 100 transistors will switch retrieving a weight from the memory.

There is also an energy cost associated with the sigmoid, but it is insignificant since it only happens once for every 10,000 synaptic computations.

There is also an energy cost associated with delivering an activation to its synaptic destinations. This is probably a small percentage of the synaptic computation cost if connectivity is predominantly local, and each processor handles a large number of neurons.

There is also an energy cost associated with learning. Hebbian learning [32], $dw_{ij} = \epsilon s_i s_j$, could be implemented with a single multiplier, and could be kept to a fraction of the synaptic computation cost by slow adaptation.

So it looks like synaptic computation and weight retrieval dominate the energy cost in a digital network, and that this cost can be brought within a factor of a few hundred of biology (see Figure 20).

10.2 Sparsely connected nets

If the network topology is not physically expressed in the hardware, then it must be folded onto the hardware topology. This folding results in time multiplexed utilization of resources, and requires labeling of interconnections. Unless the network topology can be reconstructed algorithmically, data flowing in the system will have to be labeled. Another way of saying this is that as soon as the network becomes sparse, state information in the network (neural activations and synaptic weights) must be accompanied by an address. In the limit, this would increase memory storage requirements by a factor of 37 (since $2^{37} = 10^{11}$), which, though onerous, is far better than implementing a fully connected network, which would increase memory storage requirements by a factor of 10^6.

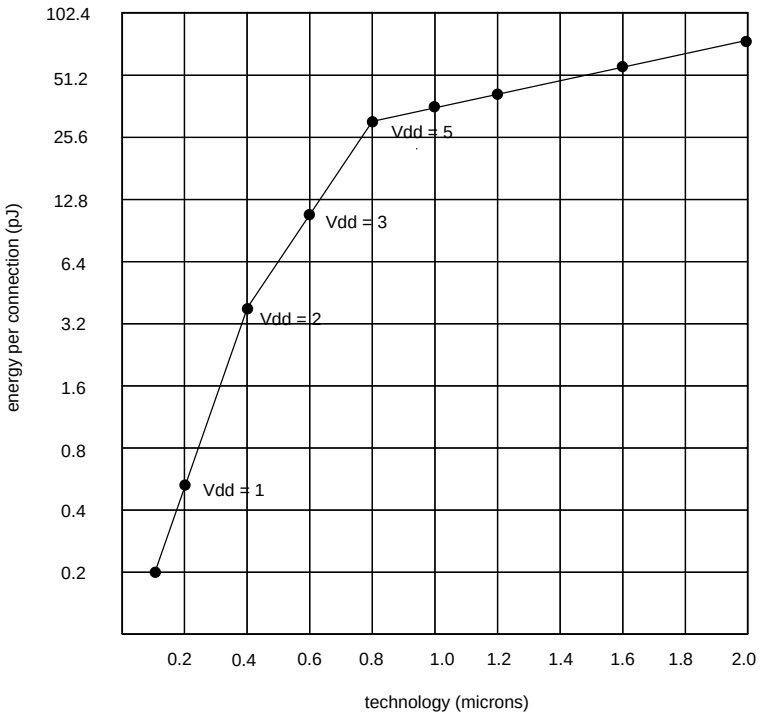


Figure 20: energy per connection vs technology

Algorithmic and explicit labeling can be combined by partitioning the network into a hierarchy of subnetworks and recursively labeling the subnets down to some level at which algorithmic labeling is introduced. For example, if the average neural fanin is 10,000, the network could be partitioned into fully connected sets of 10^4 neurons, with 10^8 synaptic weights in a set.

If we can avoid labeling synapses, and just label neural outputs, then communication bandwidth increases by a factor of 37, but synaptic storage stays the same. This is fortunate, since communication bandwidth is the easiest to increase. We can do this by grouping neurons in fully connected clusters. There will be a slight increase in synaptic storage required since the neurons will not in general be fully connected.

The most straightforward way to specify the interconnections among N neurons is with a matrix of N^2 elements. Suppose the network connections are random and uniformly distributed, ie the probability that any two neurons are connected is a constant P. Then the probability that a randomly selected matrix element is nonzero is also P, and the total number of nonzero weights is PN^2. If P is less than $log_2(w_{max})/(log_2(w_{max}) + log_2(N))$, it is more efficient to store the address of each neuron along with each weight than to store all the weights. That is, rather than store $(w_{ij} : i = 1, N; j = 1, N)$, store $(w_{ij}, j : w_{ij} <> 0)$. It is not necessary to store i explicitly if weights are clustered by output. However, the input neuron ids must be stored along with each weight if the network is randomly connected.

Any regularity which might exist in the pattern of network connections can be exploited to reduce the number of neuron ids which must be stored. In some cases, such as fully

	neurons	APS	synapses	CPS
[53]	-	-	10^{16}	10^{16}
[76]	10^{11}	10^{13}	10^{14}	10^{16}
Sejnowski PNS	10^{9}	10^{12}	10^{13}	10^{16}
Sejnowski CNS	10^{11}	10^{11}	10^{15}	10^{15}

Table 7: Cortical statistics

connected networks, or networks whose pattern of connectivity is the same for every neuron, id storage can be eliminated, since weights can be allocated to processors in a completely uniform way.

If any subset of the network has a regular connection pattern, then id storage for the neurons in that subnet can be eliminated. For example. in a multilayer feedforward net, if each pair of adjacent layers is fully connected, then only one id is needed in each layer to identify the entire "block" of connections. All the connections within that block can be computed as a fully connected set relative to a single base address.

The number of block ids which must be stored is equal to the number of tiles required to cover the nonzero entries in the weight matrix. There is a significant tradeoff in tiling function complexity and flexibility. The more powerful the tiling function, the higher its computational cost. The objective is to find a set of functions and a tiling of those functions which maximizes coverage and minimizes overlap, at modest computational cost.

10.3 Biological complexity

VLSI may have been oversold in its ability to provide sufficient resources for large scale network computation. It is tempting to endow VLSI with essentially limitless capabilities, so that any problem, no matter how intractable, can be solved. In neural networks, VLSI may have met its match. The major challenges are computation, communication, and memory. Since silicon is much faster, computation, and communication can be time multiplexed. Memory, however, cannot.

Table 7 gives some published estimates of performance and capacity of the human cortex. Tables 8 and 9 further refine those estimates into peripheral and central nervous system operations.

The bandwidth of a single metal wire is somewhere between 10 Mbit/sec and 1 Gbit/sec, depending on the technology. In the peripheral nervous system (PNS), the average bandwidth of a single nerve fiber is somewhere around 1 Kbit/sec. Therefore each silicon wire could carry the information of roughly 10^5 PNS neurons. In the central nervous system (CNS),

10^9	neurons
10^3	activations/sec/neuron
10^4	fanout
10^{13}	synapses
10^{12}	activations/sec
10^{16}	connections/sec

Table 8: Peripheral nervous system capacity and performance

10^{11}	neurons
1	activation/sec/neuron
10^4	fanout
10^{15}	synapses
10^{11}	activations/sec
10^{15}	connections/sec

Table 9: Central nervous system capacity and performance

the average bandwidth of a single axon is around 1 bit/sec, so each silicon wire could carry the information of roughly 10^8 CNS neurons.

The computational bandwidth of a neuron is much more difficult to quantify because it depends to an overwhelming extent on the logical complexity of the model used to describe it. The secret is to find a good match between the set of available logic elements and the algorithm to be implemented.

Table 10 suggests that a single VLSI processor could conceivably process 10^9 synapses per second, and might therefore handle 10^2 PNS neurons or 10^5 CNS neurons. 10^7 PNS processors and 10^6 CNS processors would be required. Each PNS processor would service 10^6 synapses; each CNS processor 10^9 synapses.

	PNS	CNS
processors	10^7	10^6
neurons/proc	10^2	10^5
synapses/proc	10^6	10^9

Table 10: Cortical processor architectures

tech	syn/cm^2 S^2	macs/cm^2 S^2	CPS/cm^2 S^3	cm^2/10^{15}syn S^2	cm^2/10^{16}CPS S^3
2.00μ	1e6	1e2	1e10	1e9	1e6
1.00μ	4e6	4e2	8e10	2e8	1e5
0.50μ	2e7	2e3	6e11	4e7	1e4
0.25μ	1e8	6e3	5e12	1e7	2e3
0.10μ	1e9	4e4	8e13	1e6	1e2

Table 11: Capacity and performance (scale V)

Table 11 suggests that the synaptic storage problem is four orders of magnitude more challenging than achieving adequate performance. It also suggests that achieving comparable computational bandwidth is perhaps feasible, but that comparable storage capacity will require a technological breakthrough. The capacity numbers assume only one storage cell is required per synapse. Analog synaptic cells reported so far are two orders of magnitude larger (see Table 12).

The size of an individual synaptic storage element is the major limiting factor in determining the size of a network.

Table 12 illustrates the wide range in synaptic densities among reported alternatives.

Until someone comes up with a much more area efficient analog synaptic cell, or until technology scales well below 0.1 micron, adaptive analog synapses will remain too big to scale up to biology. The only VLSI chance for large nets is 1T DRAM or 4T SRAM.

Leakage current, according to Chatterjee et al[14], is 3fA/μ^2, and is a strong function of temperature. A 0.5μ 1T DRAM storage node with an area of $2\mu^2$ has a leakage current of 6fA/bit. That's a total leakage current of 6 amps for 1e15 bits.

An important consequence of these observations is that neurochip architectures which emphasize computational bandwidth but have low synaptic storage densities will not scale well to large networks.

11 Conclusion

Digital VLSI can be used to implement a wide variety of neural networks. Both very high performance and very large scale networks can be implemented effectively, especially as

	type	$\lambda \times \lambda$	λ^2	λ^2/T	rel.area	who
1T	EPROM	8×8	64	64	1	commercial
1T	DRAM	8×8	64	64	1	commercial
4T	SRAM	12×20	240	60	4	commercial
1T	DRAM	8×16	128	128	2	MOSIS
2T	DRAM	13×31	403	200	6	MOSIS
3T	DRAM	24×24	576	192	9	MOSIS
6T	SRAM	32×36	1152	192	18	MOSIS
7T	synapse	40×60	2400	342	37	[48]
??T	ETANN	83×97	8036	??	125	[38]
42T	synapse	200×200	40000	952	625	[9]
200T	CLC	160×320	51200	256	700	[5]

Table 12: Area of various synaptic cells

technology scales down into the submicron regime. There is a tradeoff between performance and flexibility. Two significant challenges for future research are increasing synaptic storage capacity and minimizing power consumption.

The connectionist approach assumes synaptic communication can be modeled simply. The limit to the size of networks which VLSI can implement depends not only on the energy per computation and the synaptic storage density, but also on the complexity of the synaptic interaction and the neuron model.

Acknowledgements

This work was supported by the NASA Center for Aeronautics and Space Information Systems (CASIS) under grant NAGW 419.

References

[1] David H. Ackley and Geoffrey E. Hinton. A learning algorithm for Boltzmann Machines. *Cognitive Science*, 9:147–169, 1985.

[2] Aharon J. Agranat, Charles F. Neugebauer, and Amnon Yariv. A CCD based neural network integrated circuit with 64K analog programmable synapses". In *IJCNN International Joint Conference on Neural Networks*, pages II:551–555, 1990.

[3] Shingo Aizaki, Masayoshi Ohkawa, Akane Aizaki, Yasushi Okuyama, Isao Sasaki, Toshiyuki Shimizu, Kazuhiko Abe, Manabu Ando, and Osamu Kudoh. A 15ns 4Mb CMOS SRAM. In *IEEE International Solid-State Circuits Conference*, pages 126–127, 1990.

[4] Yutaka Akiyama. *The Gaussian Machine: a stochastic, continuous neural network model.* PhD thesis, Keio University, February 1990.

[5] Joshua Alspector. CLC - A cascadeable learning chip. NIPS90 VLSI Workshop, December 1990.

[6] Joshua Alspector, Joel W. Gannett, Stuart Haber, Michael B. Parker, and Robert Chu. A VLSI-efficient technique for generating multiple uncorrelated noise sources and its application to stochastic neural networks. IEEE Transactions on Circuits and Systems, 1990.

[7] Joshua Alspector, Bhusan Gupta, and Robert B. Allen. Performance of a stochastic learning microchip. In *Advances in Neural Information Processing Systems*, pages 748–760, 1989.

[8] Masakazu Aoki, Yoshinobu Nakagome, Masahi Horiguchi, Shin'ichi Ikenaga, and Katsuhiro Shimohigashi. A 16-level/cell dynamic memory. *IEEE Journal of Solid-State Circuits*, pages 297–299, April 1987.

[9] Yutaka Arima, Koichiro Mashiko, Keisuke Okada, Tsuyoshi Yamada, Atushi Maeda, Harufusa Kondoh, and Shinpei Kayano. A self-learning neural network chip with 125 neurons and 10K self-organization synapses. In *Symposium on VLSI Circuits*, pages 63–64, 1990.

[10] Algirdas Avizienis. Signed-digit number representations for fast parallel arithmetic. *IRE Transactions on Electronic Computers*, pages 389–400, September 1961.

[11] Jim Bailey and Dan Hammerstrom. Why VLSI implementations of associative VLCNs require connection multiplexing. In *IEEE International Conference on Neural Networks*, pages II:173–180, 1988.

[12] G. Blelloch and C. R. Rosenberg. Network learning on the Connection Machine. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 323–326, 1987. Milan, Italy.

[13] James B. Burr et al. A 20 MHz Prime Factor DFT Processor. Technical report, Stanford University, September 1987.

[14] Pallab K. Chatterjee, Geoffrey W. Taylor, Al F. Tasch, and Horng-Sen Fu. Leakage studies in high-density dynamic MOS memory devices. *IEEE Transactions on Electron Devices*, pages 564–575, April 1979.

[15] Alice Chiang, Robert Mountain, James Reinold, Jeoffrey LaFranchise, James Gregory, and George Lincoln. A programmable CCD signal processor. In *IEEE International Solid-State Circuits Conference*, pages 146–147, 1990.

[16] David Van den Bout and Thomas K. Miller III. TInMANN: The integer Markovian artificial neural network. In *IJCNN International Joint Conference on Neural Networks*, pages II:205–211, 1989.

[17] Sang H. Dhong, Nicky Chau-Chau Lu, Wei Hwang, and Stephen A. Parke. High-speed sensing scheme for CMOS DRAM's. *IEEE Journal of Solid-State Circuits*, pages 34–40, February 1988.

[18] M. Duranton and J.A. Sirat. Learning on VLSI: A general purpose digital neurochip. In *IJCNN International Joint Conference on Neural Networks*, 1989.

[19] Silvio Eberhardt, Tuan Duong, and Anil Thakoor. Design of parallel hardware neural network systems from custom analog VLSI 'building block' chips. In *IJCNN International Joint Conference on Neural Networks*, pages II:183–190, 1989.

[20] Brian R. Gaines. Stochastic computing. In *Spring Joint Computer Conference*, pages 149–156, 1967.

[21] Brian R. Gaines. Uncertainty as a foundation of computational power in neural networks. In *IEEE First International Conference on Neural Networks*, pages III:51–57, 1987.

[22] Simon C. J. Garth. A chipset for high speed simulation of neural network systems. In *IEEE First International Conference on Neural Networks*, pages III:443–452, 1987.

[23] James A. Gasbarro and Mark A. Horowitz. A single-chip, functional tester for VLSI circuits. In *IEEE International Solid-State Circuits Conference*, pages 84–85, 1990.

[24] Hans Peter Graf and Don Henderson. A reconfigurable CMOS neural network. In *IEEE International Solid-State Circuits Conference*, pages 144–145, 1990.

[25] H.P. Graf, W. Hubbard, L.D. Jackel, and P.G.N. deVegvar. A CMOS associative memory chip. In *IEEE First International Conference on Neural Networks*, pages III:461–468, 1987.

[26] H.P. Graf, L.D. Jackel, R.E. Howard, B. Straughn, J.S. Denker, W. Hubbard, D.M. Tennant, and D. Schwartz. VLSI implementation of a neural network memory with several hundreds of neurons. In *Neural Networks for Computing, Snowbird, Utah 1986*, pages 182–187. American Institute of Physics, 1986.

[27] A. J. De Groot and S. R. Parker. Systolic implementation of neural networks. In *SPIE, High Speed Computing II*, volume 1058, January 1989.

[28] Dan Hammerstrom. A VLSI architecture for high-performance, low-cost, on-chip learning. In *IJCNN International Joint Conference on Neural Networks*, pages II:537–544, 1990.

[29] Yoshihisa Harata, Yoshio Nakamura, Hiroshi Nagase, Mitsuharu Takigawa, and Naofumi Takagi. A high-speed multiplier using a redundant binary adder tree. *IEEE Journal of Solid-State Circuits*, pages 28–34, February 1987.

[30] Shigeyuki Hayakawa, Masakazu Kakumu, Hideki Takeuchi, Katsuhiko Sato, Takayuki Ohtani, Takeshi Yoshida, Takeo Nakayama, Shigeru Morita, Masaaki Kinugawa, Kenji Maeguchi, Kiyofumi Ochii, Jun'ichi Matsunaga, Akira Aono, Kazuhiro Noguchi, and Tetsuya Asami. A 1uA retention 4Mb SRAM with a thin-film-transistor load cell. In *IEEE International Solid-State Circuits Conference*, pages 128–129, 1990.

[31] Raymond A. Heald and David A. Hodges. Multilevel random-access memory using one transistor per cell. *IEEE Journal of Solid-State Circuits*, pages 519–528, August 1976.

[32] Donald O. Hebb. *The Organization of Behavior.* Wiley, 1949.

[33] John L. Hennessy and David A. Patterson. *Computer Architecture A Quantitative Approach.* Morgan Kaufmann Publishers, 1989.

[34] G.E. Hinton, T.J. Sejnowski, and D.H. Ackley. Boltzmann Machines: Constraint satisfaction networks that learn. Technical Report CMU-CS-84-119, Carnegie-Mellon University, May 1984.

[35] Geoffrey E. Hinton. Deterministic Boltzmann learning performs steepest descent in weight-space. *Neural Computation*, 1:143–150, 1989.

[36] Yuzo Hirai, Katsuhiro Kamada, Minoru Yamada, and Mitsuo Ooyama. A digital neurochip with unlimited connectability for large scale neural networks. In *IJCNN International Joint Conference on Neural Networks*, pages II:163–169, 1989.

[37] David A. Hodges and Horace G. Jackson. *Analysis and Design of Digital Integrated Circuits.* McGraw-Hill, 1983.

[38] Mark Holler, Simon Tam, Hernan Castro, and Ronald Benson. An electrically trainable artificial neural network (ETANN) with 10240 "floating gate" synapses. In *IJCNN International Joint Conference on Neural Networks*, pages II:191–196, 1989.

[39] J.J. Hopfield. Neural networks and physical systems with emergent collective computational properties. *Proceedings of the National Academy of Sciences, USA*, 79:2554–2558, April 1982.

[40] Masahi Horiguchi, Masakazu Aoki, Yoshinobu Nakagome, Shin'ichi Ikenaga, and Katsuhiro Shimohigashi. An experimental large-capacity semiconductor file memory using 16-levels/cell storage. *IEEE Journal of Solid-State Circuits*, pages 27–33, February 1988.

[41] Robert R. Johnson. Multichip modules: Next-generation packages. *IEEE Spectrum*, pages 34–48, March 1990.

[42] Max Stanford Tomlinson Jr., Dennis J. Walker, and Massimo A. Sivilotti. A digital neural network architecture for VLSI. In *IJCNN International Joint Conference on Neural Networks*, pages II:545–550, 1990.

[43] Howard Kalter, John Barth, John Dilorenzo, Charles Drake, John Fifield, William Hovis, Gordon Kelley, Scott Lewis, John Nickel, Charles Stapper, and James Yankosky. A 50ns 16Mb DRAM with a 10ns data rate. In *IEEE International Solid-State Circuits Conference*, pages 232–233, 1990.

[44] Teuvo Kohonen. *Self-Organization and Associative Memory.* Springer-Verlag, 1984.

[45] S. Y. Kung and J. N. Hwang. Digital VLSI architectures for neural networks. In *IEEE International Symposium on Circuits and Systems*, pages 445–448, 1989.

[46] S.Y. Kung and J.N. Hwang. Parallel architectures for artificial neural nets. In *IEEE International Conference on Neural Networks*, pages II:165–172, 1988.

[47] S.Y. Kung and J.N. Hwang. Ring systolic designs for artificial neural nets. In *INNS First Annual Meeting*, pages 390–, 1988.

[48] Bang W. Lee, Ji-Chien Lee, and Bing J. Sheu. VLSI image processors using analog programmable synapses and neurons. In *IJCNN International Joint Conference on Neural Networks*, pages II:575–580, 1990.

[49] Weiping Li. *The Block Z transform and applications to digital signal processing using distributed arithmetic and the Modified Fermat Number transform.* PhD thesis, Stanford University, January 1988.

[50] Weiping Li and James B. Burr. Parallel multiplier accumulator using 4-2 adders. US patent pending, Application number 088,096, filing date August 21, 1987.

[51] Weiping Li and James B. Burr. An 80 MHz Multiply Accumulator. Technical report, Stanford University, September 1987.

[52] P. Mars and W. J. Poppelbaum. *Stochastic and deterministic averaging processors.* IEE, 1981.

[53] Carver Mead. Neuromorphic Electronic Systems. *Proceedings of the IEEE*, pages 1629–1636, 10 1990.

[54] Carver Mead and Lynn Conway. *Introduction to VLSI Systems.* Addison-Wesley, 1980.

[55] A. Moopenn, A.P. Thakoor, T. Duong, and S.K. Khanna. A neurocomputer based on an analog-digital hybrid architecture. In *IEEE First International Conference on Neural Networks*, pages III:479–486, 1987.

[56] Takayuki Morishita, Youichi Tamura, and Tatsuo Otsuki. A BiCMOS analog neural network with dynamically updated weights. In *IEEE International Solid-State Circuits Conference*, pages 142–143, 1990.

[57] 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, 1988.

[58] L. W. Nagel. SPICE2: A computer program to simulate semiconductor circuits. Technical report, University of California, Berkeley, May 9 1975. Memo ERL-M520.

[59] J. Ouali and G. Saucier. Silicon compiler for neuro-ASICs. In *IJCNN International Joint Conference on Neural Networks*, pages II:557–561, 1990.

[60] Carsten Peterson and Eric Hartman. Explorations of the mean field theory learning algorithm. In *Neural Networks*, volume 2, pages 475–494. Pergamon Press, 1989.

[61] Dean A. Pomerleau, George L. Gusciora, David S. Touretzky, and H.T. Kung. Neural network simulation at Warp speed: How we got 17 million connections per second. In *IEEE International Conference on Neural Networks*, pages II:143–150, 1988.

[62] Scott R. Powell and Paul M. Chau. Estimating power dissipation in VLSI signal processing chips: the PFA technique. In *VLSI Signal Processing IV*, pages 251–259, 1990.

[63] Jack Raffel, James Mann, Robert Berger, Antonio Soares, and Sheldon Gilbert. A generic architecture for wafer-scale neuromorphic systems. In *IEEE First International Conference on Neural Networks*, pages III:501–513, 1987.

[64] Mike Rudnick and Dan Hammerstrom. An interconnect structure for wafer scale neurocomputers. In David Touretzky, Geoffrey Hinton, and Terrence Sejnowski, editors, *Connectionist Models Summer School*, pages 498–512. Carnegie Mellon University, 1988.

[65] J.P. Sage, K. Thompson, and R.S. Withers. An artificial neural network integrated circuit based on MNOS/CCD principles. In *Neural Networks for Computing, Snowbird, Utah 1986*, pages 381–385. American Institute of Physics, 1986.

[66] Mark Santoro and Mark Horowitz. A pipelined 64X64b iterative array multiplier. In *IEEE International Solid-State Circuits Conference*, pages 35–36, February 1988.

[67] Mark Santoro and Mark Horowitz. SPIM: A pipelined 64X64-bit iterative multiplier. *IEEE Journal of Solid-State Circuits*, pages 487–493, April 1989.

[68] Mark R. Santoro. *Design and Clocking of VLSI Multipliers.* PhD thesis, Stanford University, October 1989.

[69] Terrance J. Sejnowski and Charles R. Rosenberg. NETtalk: A parallel network that learns to read aloud. Technical Report JHU/EECS-86/01, Johns Hopkins University, 1986.

[70] D. T. Shen and A. Weinberger. 4-2 carry-save adder implementation using send circuits. In *IBM Technical Disclosure Bulletin*, pages 3594–3597, February 1978.

[71] M. Sivilotti, M. Mahowald, and C. Mead. Realtime visual computations using analog CMOS processing arrays. In *Proceedings of the 1987 Stanford Conference*, 1987.

[72] Chris J. Terman. User's guide to NET, PRESIM, and RNL/NL. Technical Report VLSI 82-112, Massachusetts Institute of Technology, 1982.

[73] L. Waller. How MOSIS will slash the cost of IC prototyping. *Electronics*, 59(9), March 3 1986.

[74] Shlomo Waser and Michael J. Flynn. *Introduction to Arithmetic for Digital Systems Designers*. CBS College Publishing, 1982.

[75] Takumi Watanabe, Yoshi Sugiyama, Toshio Kondo, and Yoshihiro Kitamura. Neural network simulations on a massively parallel cellular array processor: AAP-2. In *IJCNN International Joint Conference on Neural Networks*, pages II:155–161, 1989.

[76] Carol Weiszmann, editor. *DARPA Neural Network Study, October 1987 - February 1988*. AFCEA International Press, 1988.

[77] Neil Weste and Kamran Eshraghian. *Principles of CMOS VLSI Design - A Systems Perspective*. Addison-Wesley, 1985.

[78] William Wike, David Van den Bout, and Thomas Miller III. The VLSI implementation of STONN. In *IJCNN International Joint Conference on Neural Networks*, pages II:593–598, 1990.

[79] Kazuo Yano, Toshiaki Yamanaka, Takashi Nishida, Masayoshi Saito, Katsuhiro Shimohigashi, and Akihiro Shimizu. A 3.8-ns CMOS 16x16-b multiplier using Complementary Pass-Transistor Logic. *IEEE Journal of Solid-State Circuits*, pages 388–394, April 1990.

[80] Moritoshi Yasunaga, Noboru Masuda, Mitsuo Asai, Minoru Yamada, Akira Masaki, and Yuzo Hirai. A wafer scale integration neural network utilizing completely digital circuits. In *IJCNN International Joint Conference on Neural Networks*, pages II:213–217, 1989.

[81] Moritoshi Yasunaga, Noboru Masuda, Masayoshi Yagyu, Mitsuo Asai, Minoru Yamada, and Akira Masaki. Design, fabrication and evaluation of a 5-inch wafer scale neural network LSI composed of 576 digital neurons. In *IJCNN International Joint Conference on Neural Networks*, pages II:527–535, 1990.