Chapter 1

Abacus Chip

This chapter describes the design, implementation, and testing of the Abacus-1 chip in considerable detail. After reading this section, the reader should understand the issues involved in the design of a high-speed SIMD computer. The discussion focuses on the implementation, assuming that the broad architectural issues were previously decided. Of course in reality the design did not proceed unidirectionally from architecture to implementation; low level issues significantly affected the architecture.

The design goals of the Abacus-1 chip were not simply to obtain the highest possible performance, but more importantly to investigate issues of high-speed SIMD computers. Therefore, performance improvements at the cost of higher complexity and lower clock rate are not desirable. Further, the high speed must transfer over to the system level, so the chip design must handle inter-chip synchronization effectively. Proceeding up the hierarchy, the lessons learned from the design must apply to practical systems. A critical requirement of real systems is the capability of expanding the memory allocated to each processing element. Finally, the system must support high speed (at least real-time) I/O of image data.

A large collection of on-chip subsystems is required to satisfy these goals. The subsystems can be divided into three broad categories: the processing element core that performs the computation; the off-chip interface that delivers network, instruction, and data streams to the core; and the control logic that binds the two together. This chapter addresses each of these categories in turn.

1.1 Processing Element Core

1.1.1 Memory

Design Processor memory size is a critical design parameters as it is the main determinant of the overall processor size and therefore the number of PEs per chip. Too little on-chip memory, and machine performance becomes limited by off-chip memory bandwidth; too much memory and the amount of processing power on a chip drops. Thus, the memory size should be carefully chosen based on extensive simulations.

In the case of the Abacus-1 chip, the choice was mostly guided by intuition. There is a fairly small number of feasible sizes, between 32 and 1024 bits. The low end of the range is inefficient as, much of the memory is consumed by configuration and temporary bits. The 32-bit configuration, in vertical mode, cannot perform any g algorithm that requires more than two 16-bit data items. The 1024-bit configuration, in horizontal mode, allocates 1 Kword per pixel site, which is far more memory than needed. Two other disadvantages of larger memories are a longer access time, and more address bits, which translates directly in higher instruction bandwidth, already one of the more challenging board-level design tasks. The Abacus-1 design has 64 bits of memory per PE.

Implementation The 64-bit register file is arranged as two banks of eight rows and four columns. Each cell is a standard six-transistor SRAM cell. The bit and $\overline{\text{bit}}$ lines are used single-ended to provide two read ports, allowing four bits to be read and two written in each cycle.

The bitlines are precharged to $V_{dd} - V_T$ to speed pulldown time and reduce power dissipation. Precharging is done through the column select transistors, further reducing power by charging only those bitlines that are about to be read.

The SRAM cell is susceptible to false writes, especially because the bitlines are typically not charged to opposite values. Thus, the high-capacitance bitlines, charged to opposing values during the read phase, can act as a line driver, overwriting cells not selected by the column selects. This problem is avoided by shorting the bit and $\overline{\text{bit}}$ lines during the precharge interval, even if those lines are charged. In other words, all bitline pairs are shorted, but only the ones to be read are charged.

One of the key space-saving ideas was the integration of the five control registers and of the network interface into the SRAM array. The initial design included fully static master-slave flip-flops, multiplexers to connect them to read busses, and separate control wires. This auxiliary circuitry substantially increased the non-ALU PE area. In the current implementation, edge cells in the SRAM array, together with buffers that isolate the internal data storage node, are used as control registers. In the case of the network and data plane registers, the multiplexing occurs inside the SRAM cell.

1.1.2 ALU

Design The main design issue of the ALU is its range of computations. The initial ALU design was a 4:1 multiplexer that could compute all 16 two-input one-output functions. Computation of a full binary one-bit addition therefore required five cycles, emulating all five 2-input gates.

The limited complexity is due to available data sources and sinks. A single-ported memory bank can produce only two bits for reading, and can absorb only a one bit result. If operations such as addition are required, the extra input bit must come from another

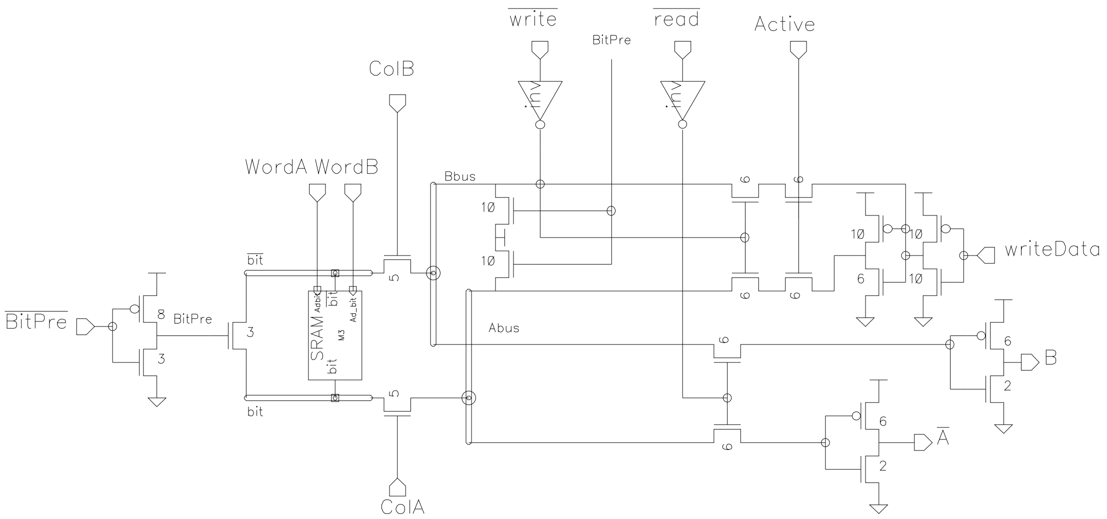

Figure 1-1: Memory read and write path (one of the banks)

source such as an auxiliary register or the communication network.

Previous bit-serial computers incorporated dedicated adders and carry registers. Abacus does not follow this approach because of the additional area and because the circuitry is only useful in vertical mode. A dedicated full adder circuit would be useful in horizontal mode, assuming its outputs could be steered appropriately, but it requires three input bits while the memory only produces two per cycle. Our solution was based on the realization that the memory required much more area than the ALU. Splitting the memory into two banks doubles the memory bandwidth and adding another ALU doubles the peak computing power. If the ALU requires one tenth of the memory area, the twin-bank, twin-ALU (TBTA) design occupies 1.1 times the area, but has 2.0 times the performance, for an overall advantage of 1.8 over the single ALU. This appears to be a substantial improvement.

A significant penalty occurs because the chip is pad-limited. The inner pad-ring is arranged in a 56 by 44 rectangle. Without the 20 extra instruction bits (corresponding to 10 double-cycled pads) the pad ring could be reduced to a 51 by 44 rectangle. Assuming a 170 micron pad pitch, the area decrease is 6.4 mm^2. Since the entire PE array occupies 38.4 mm^2, the extra instruction bits cost an additional 17% overhead. As long as the design remains pad limited, this effect will grow worse as technologies scale down, mostly because the pad size stays constant while the PE density improves. Thus, any design decision that increases the number of chip pads should be considered carefully. This is an important example of how architectural analysis must be done at all levels of design in order to be accurate.

The algorithmic advantage is more difficult to quantify. Each 3-input ALU can perform an arbitrary 3-input operation each cycle, which could take a 2-input ALU at least 3 (and as

Circuit Element	SBSA 1 ALU, 1 bank		SBSA 2 ALU, 2 bank	
	Quant	**Area**	**Quant**	**Area2**
Memory Array	1	205	1	205
Column Mux	8	20	8	20
Read/Write Logic	1	18	2	36
Aux Logic	1	64	1	64
ALU	1	38	2	76
Network	1	52	1	52
Total		397		453

Table 1.1: Table 1 Silicon area requirements of the single-banked, single-ALU design and the double-banked, double-ALU design. As expected silicon area increases by only 15%.

many as 5) cycles. To see this, consider the an arbitrary function of three boolean variables Z = f(A,B,C). Using the Shannon expansion, Z can be expressed as

$$Z = (\overline{A} \cdot f_{A=0}(B,C)) + A \cdot *f_{A=1}(B,C)) \tag{1.1}$$

where the two subfunctions are obtained from the truth table entries corresponding to the cases where A is 0 and 1 respectively. Most useful functions are actually simpler. For example, consider the the multiplexing operation `Z = (!A & B) | (A & C)`.[1] For this function, $f_{A=0}(B,C)$ is simply B, and steps 1 and 3 of Figure **??** can be eliminated.

```
T1  =  f0(B,C)
T1  =  T1 & !A
T2  =  f1(B,C)
T2  =  T2 & A
Z   =  T1 + T2
```

Figure 1-2: Evaluation of an arbitrary 3-input function by a two-input ALU. Two auxiliary memory locations are required.

Although converting to two 3-input ALUs improves the peak bit-manipulation performance by a factor of ten. But several effects limit this improvement. First, the addressing is limited to sharing certain bits between the two banks. Second, these five-cycle operations are relatively rare. Third, data dependencies require waiting for bits to arrive from the network.

Table **??** shows the number of cycles required per arithmetic operation, including reconfig-

[1] This function is used frequently when some bits must be conditionally updated. The idiom is `Z := mux(flag, Z, newZ)`.

uration overhead. The ratio is about 1.75. Of course, the proper instruction mix for real applications should be evaluated for true improvements, but this analysis suffices to bound performance improvement in the range of 1.25 to 2.1.

Operation	SBSA (cycles)	TBTA (cycles)
Add	9	4
Shift	4	3
Grid Move	7	6
Accumulate	10	4
Average	7.5	4.3

Table 1.2: Number of cycles required for common operations in horizontal mode. The count includes reconfiguration cost.

There is another subtle effect of a more complicated ALU. A 2-input ALU can require up to two temporary storage locations to evaluate most 3-input functions. When operating near the memory capacity, two memory spills can have a substantial effect on performance.

We can now evaluate the overall advantage of the twin ALU design. The area cost is dominated by the pad ring increase, and is a factor of 1.35. The performance improvement is 1.75. Thus, the overall advantage of the design is only about 30%. This advantage will increase slightly as compiler technology improves and as memory accesses become more expensive.

Implementation. Each ALU is implemented as a precharged 8-to-1 multiplexer that uses three data bits to select one of eight instruction bits. Initial design used predecoded cells but this was slower and larger due to the non-regularity. This design does have four transistors in the pulldown path, but the timing and capacitances are designed to minimize the body effect and reduce charge-sharing problems.

Immediate Constant. The ALU also incorporates an additional circuit that allows the architecture to support effective constant distribution. Without this feature, there is no clean method of distributing a run-time value generated in the scalar unit to the array. Indeed the only way of transferring such a value would be to send it via the image IO interface.

Instead, the output of the left ALU passes through a NAND gate controlled by a global wire common to each column of PEs. In normal operation, these 32 global "literal" wires are held at 1. When a constant is to be loaded, the instruction decoder generates a `set` ALU op code, and loads the literal bus with the logical inverse of the constant. Thus, in every column whose bus is at 1, the result will be forced to a 0, as desired.

1.1.3 Network

Design The two questions in network design are where the PE obtain its network data and how does it route data around itself. Two alternatives of routing data are shown in Figure **??**. The first, the two-channel crossbar, allows messages to pass in a wide variety of directions. For example, the PE can connect the north port to the east port while simultaneously connecting the west port to the south. The second is far less flexible, and can only make one connection.

The original network design used four independent switches in a grid topology. Each switch had a dedicated control register. The independent switch control allowed many cluster configurations.

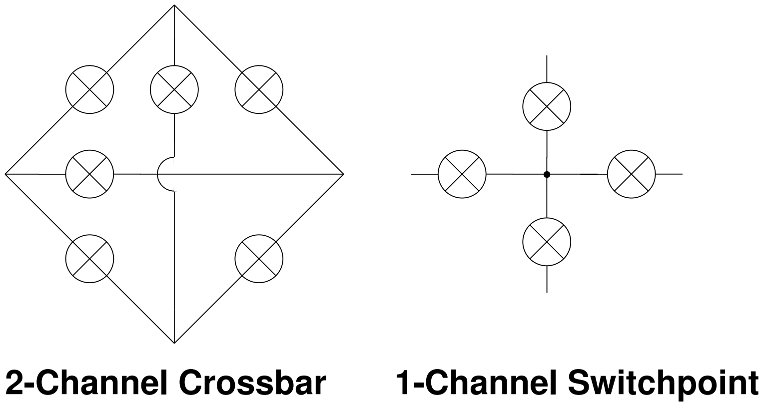

Figure 1-3: Various network types. The Abacus type is of the single channel, but even more restrictive in that each PE can turn on only one switch.

The price of this flexibility is additional registers and increased reconfiguration time. If a register is allocated to each switch, the four registers represent 5% of the total addressable registers.. More importantly, reconfiguration time doubles. Even with dual ALUs (which were added primarily to speed reconfiguration), two cycles are required to change topology. Since reconfiguration may be required after each arithmetic operation, and since most operations require only two or three cycles, the additional 1 cycle penalty is a performance hit of approximately 20%.

Reconfiguration time can be reduced at a slight loss in flexibility. Since information flow in arithmetic operations is usually along the MSB/LSB path, configuration in a line is usually sufficient. The design implemented in Abacus-1 incorporates a decoder to select only one of the four switches, and a *connect* configuration bit that controls whether any of the switches are activated at all. In sum, the addressing cost of the two approaches is not substantially different: three registers instead of four. But now, changing reconfiguration directions requires only one cycle, and conditional bypass, another frequent operation, also requires only one cycle instead of two.

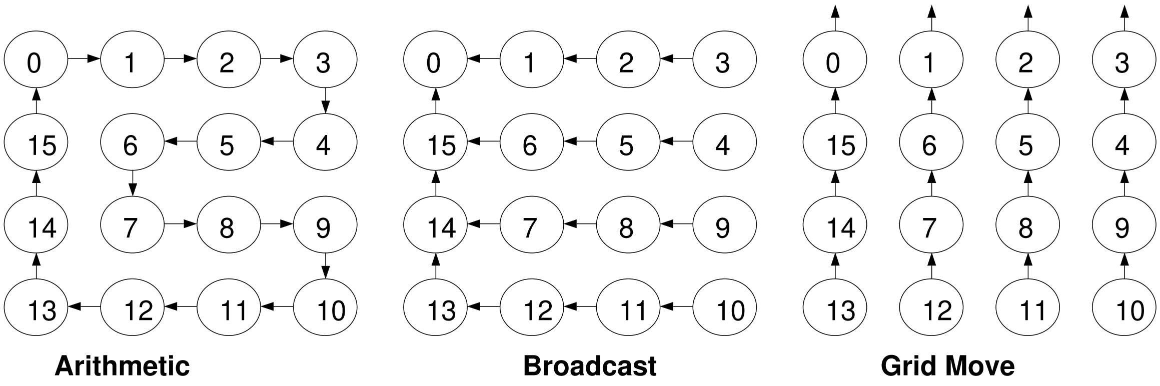

Figure 1-4: Various network configurations. Arithmetic, bit broadcast, and grid move modes are shown. For example, broadcast of a bit to all 16 cluster elements can be done with only 8 ($2\sqrt{N}$) switch delays if the clustered is configured as a fanout-4 tree.

Implementation The on-chip interprocessor communication network consists of a precharged output node and four isolating switches. The node is a simple precharged inverter whose input is driven by the network output register. The four switches are NMOS transistors driven by a 2:4 precharged decoder.

Communication with signaling times longer than a cycle time is performed by controlling network precharge via software. As long as precharge is turned off, processors connected to a bus node continue to discharge it.

Propagation delay across a series of pass transistors is usually quadratic in the number of devices. This delay is greatly reduce by a local accelerator circuit at each node which regeneratively pulls a node to ground as soon as the node voltage drops by a transistor threshold. The circuit is a dynamic NORA style circuit and is well suited to the precharged operation of the network. We found it to be several times faster than an implementation based on a complementary inverter. At the nominal process corner, simulations show that a bit propagates through 18 switches and long network wires in one 8 ns cycle.

This circuit has a noise margin of V_T, so careful layout was done to reduce coupling capacitance to the pre-discharged node. Further, the chip is provided with on-chip bypassing and over 200 power and ground pins to reduce dI/dt noise.

1.1.4 Data/IO Planes

The architecture supports a background loading mechanism for both the external memory data and image data. Data is shifted through the PE array without interfering with computation. One memory location in each PE is dedicated as the data plane (DP) registers. All DP registers in a column are connected in a 32-element shift register. This shift register is clocked by a global signal. The PE addresses the DP register as a normal SRAM cell,

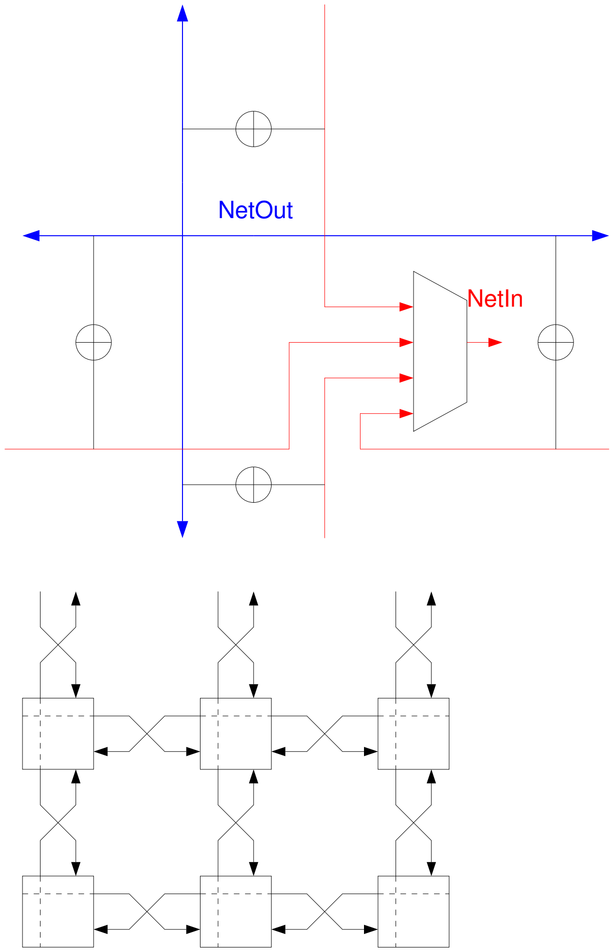

Figure 1-5: Network Block Diagram. Magnified view and grid view.

and is not aware of the alternate write path.

1.1.5 Global OR operation

For this mode, all PEs must connect their networks together. The most efficient way is to connect the leftmost column PEs vertically and the rest of the PEs horizontally. We expect 4 cycles of latency for each chip to evaluate, another for the PAL, and another for the controller PAL. Approximately 8 cycles of latency for a global OR.

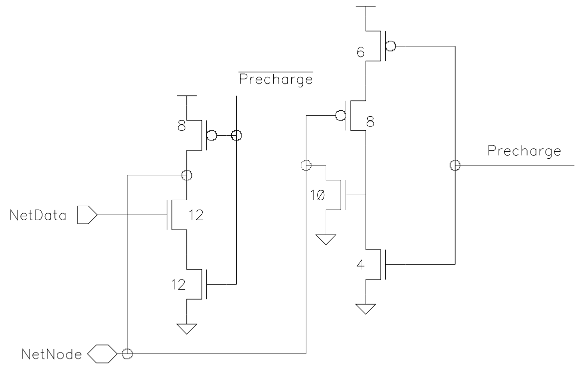

Figure 1-6: Network accelerator in the NORA circuit style.

1.1.6 Layout Summary

The simplicity of each PE resulted in a compact, 83 micron by 452 micron implementation in a 2.6 micron metal pitch technology.

Subsystem	Area ($K\mu^2$)	Percentage
Memory	22.9	61.0
ALU	6.3	16.8
Network	6.2	16.6
Misc PE	1.2	3.1
Data Plane	1.0	2.6

Table 1.3: PE Area resources

This data differs from the earlier table in that configuration registers are now allocated to the resource they serve instead of to the memory array. For example, there are four control registers (2 select, 1 connect, 1 data) dedicated to the network. Similarly, two are dedicated to the I/O and data planes.

1.2 On-Chip Signal Distribution

A previously unanticipated design task was the challenge of distributing control signals to the PEs. As discussed in section [timing], there are four phases in the clock cycle, and each phase must complete before the other can begin. For example, all word lines in the register file must be unasserted while bitlines are being precharged, and any overlap may cause memory cells to be overwritten. This nonoverlap is guaranteed by separating the

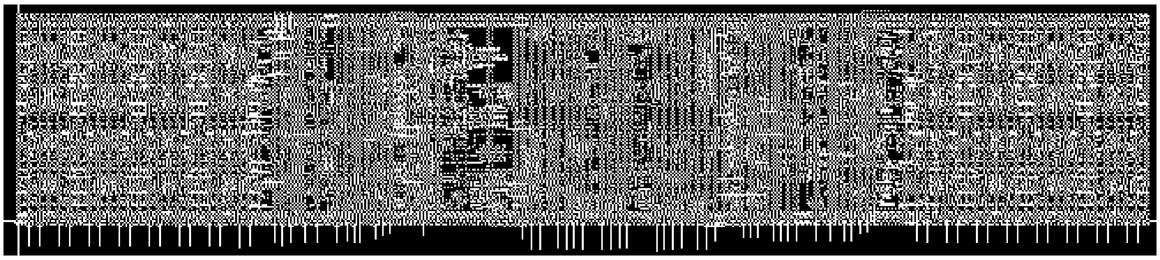

Figure 1-7: PE Layout with metal-2 shown.

Figure 1-8: PE Layout with metal-2 removed.

clock phases sufficiently to account for the worst case rise/fall times. Signal transitions that occur in the same phase but must follow each other are separated by inserting an inverter or two as delay elements.

There are two problems with this approach. Given 8 ns cycle time of the PE, each of the four phases has only 2 ns to rise, stay level, and fall. This means that even if the level time is 1 ns long, each rise/fall time must be 500 ps or less[2]. This is an aggressive RC product, given the long and narrow signal wires, and the high gate loading of the wire. The usual technique of simply widening the signal wires cannot be used due to the wire density. The second problem is that even if the four phases can be safely separated, non-overlap within the cycle by using inverters as delay elements may not work because the variation in wire loading may dwarf inverter delays. Both of these problems can be addressed by local buffering of the global signals.

1.2.1 Load Equalization

Local buffering both reduces the load on each wire and decreases the variation between signal arrival times due to unequal loads. This section describes the calculations used to obtain the load values, wire widths, and row buffer sizes.

[2]The system cycle time is currently limited by instruction wire uncertainties. For example, the bit-precharge pulse completes operation in 250 picoseconds, but the rise and fall times vary by over 300 psec.

The other part of the solution was to widen the signal traces. Although the wire capacitance increases, the lowered resistance decreases the effect of the load capacitance. Although the "optimal" width can be determined (and will be, given more writing time), in practice gate loads are spaced too closely for optimal width.

The downside of widening the traces is the increased variability of wiring capacitance. Although process variation accounts for comparable capacitance differences (6% to 7% between nominal and maximum cases for both gate and wire capacitances), more of the total capacitance varies because the wire runs over different material. Also, wire separation decreases, which increases variability due to data-dependent capacitive coupling. Both of these effects could in principle be computed by careful extraction and the use of 3-D Poisson field solvers. until the last minute.

1.2.2 Layout Issues: The Trench

The Abacus-1 design had to deliver 76 signals across a 1 cm chip with very little skew. Although the array is tolerant of gradual signal skew across the part since communication is purely local. Problems arise in two cases: intra-PE signal skew and interfaces to logically synchronous zones at at the pad ring. The task was decomposed into two parts: delivering the signals to the horizontal row drivers and then their outputs. Vertical signal delivery was not an issue due to the low resistivity of the metal 3 layer. The design easily tolerated high uniform latency in instruction distribution so the row buffers could be composed of several scale-up stages. As a result, the loads presented to the vertical wires were that of minimum sized inverters, and therefore dwarfed by wiring capacitance. The vertical wires were metal-3, 3.2 microns wide, and spaced by 3.3 microns. The nominal capacitance of a 7 mm vertical wire is 2.5 pF, and the nominal resistance is 75 ohms. The distributed RC delay of this wire is only 184 picoseconds.

The problem arises after the signals have been delivered on the vertical wires. The row buffers are rather large, and there are 152 of them packed into one PE height, 76 for each side. This occupies a significant area, and spans 1.1 mm! Fully one-sixth of the core area is dedicated to these buffers. To connect a row buffer at the extreme edge of the trench to the vertical wire requires a horizontal wire of 500 microns. This horizontal load almost doubles the signal capacitance, and therefore skew. Also, a little bit of horizontal skew is introduced, as the row buffer output wires end up differing in length, depending on the buffer position. [3]

1.2.3 Timing

The processor timing is driven by the basic design choice of double-cycling the register file to perform a read and a write operation in one cycle. The cycle starts with the instruction

[3] Notice that the routing channel alone is 500 microns. This means that row buffers had better drive only one row of signals. Otherwise, distribution of signals to other rows will require another such routing channel on each half of the array, boosting the trench width to 1500 microns!

clock, which clocks out the decoded instruction bits. A little later, the bitline precharge signal is asserted. This signal enables precharge transistors onto the A and B busses, charging them, and the bitlines that are about to be read. The column selects must therefore be stable at their read phase values during this phase. Also, the word lines must be off to avoid overwriting memory cell contents. The word lines are gated by the `ClearRow` signal, which safely surrounds the precharge pulse.

After the bitline is unasserted, the read phase executes. The word lines are enabled, allowing the selected SRAM cells to pull down their respective bitlines, and through the column selects, to pull down the A and B lines. When the read cycle is over, the contents of the A and B busses are latched on a simple NMOS dynamic latch. The ALU evaluate cycle begins next (the ALU evaluate is the read signal delayed by one inverter). All inputs to the ALU were guaranteed high for long enough during its precharge phase (since the A and B busses were precharged). The ALU outputs are stable until the next read cycle. During the ALU evaluate the bitlines are equilibrated an precharged. The address lines must be 0. Actually, while they're gated off, they're transitioning to the write addresses.

When the ALU cycle completes, the write cycle begins. The write is a double-sided write and occurs through the write enables, the column selects, and the SRAM pass transistor, so it's fairly slow. Actually, once the coupled inverters are flipped to their future state, the write completes slightly into the next cycle.

The network precharge/evaluate cycle is the write pulse. As soon as the write is over, the network evaluates. The dynamic decoder evaluates and selects one of the four switches. Simultaneously, the X node is pulled down (or not, based on the NetData register). Thus, for neighbor to neighbor communication, the just written bit has the combined time of the bit precharge period and the read period to propagate to the neighbor and pull down the bitline.

1.3 External Interfaces

1.3.1 System Synchronization

Careful synchronization is essential to a high-speed SIMD architecture. On the Abacus PE board the system clock is distributed in differential ECL to each PE chip. Propagation delays are matched by snaking traces so that trace lengths are equalized. To compensate for skew introduced by process variations in the on-chip clock distribution, each chip phase locks its internal clock to the received clock. Phase locking is performed by a delay locked loop (DLL) based on a mostly–digital delay line with a fine-tuning analog sectioqn. The DLL uses the simple and accurate arbiter circuit shown in Figure **??**. The arbiter has a perfectly symmetric 30 ps uncertainty window at nominal process.

The next design issue is the choice of which of the several signals should be synchronized across chips. There are several internal clocks: instruction register, PE register read, network precharge, etc. We choose to synchronize to the clock that drives the mesh output pads. As a result, off- chip communication is synchronized system-wide.

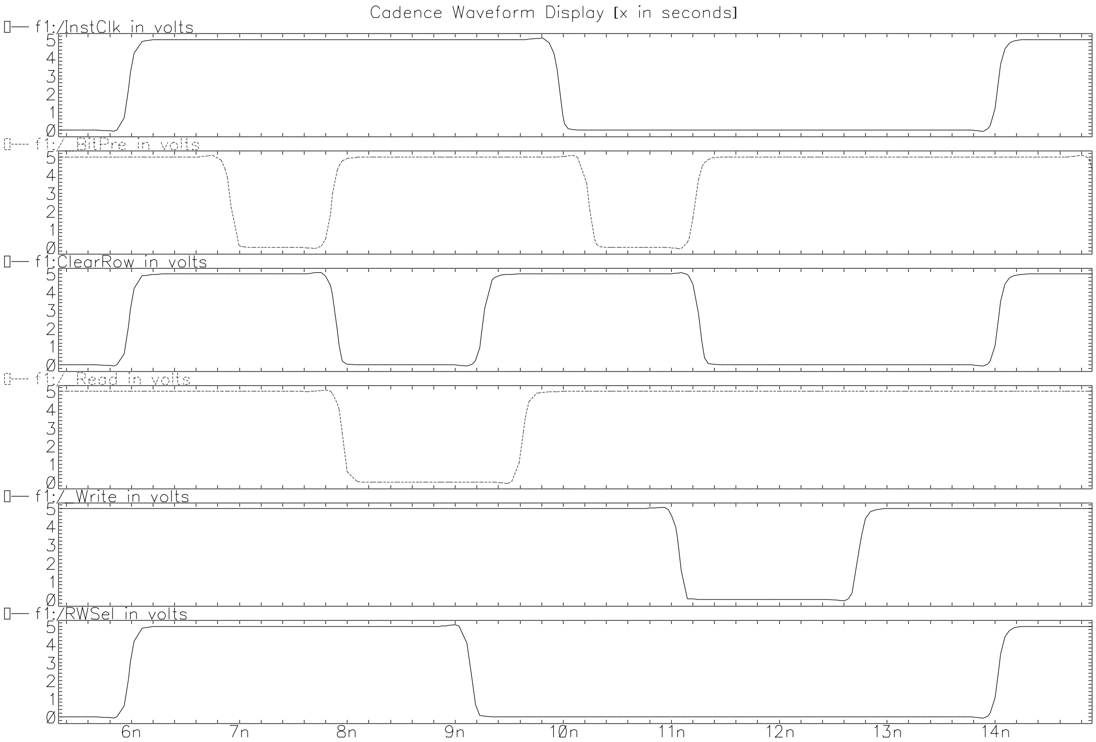

Figure 1-9: PE Timing.

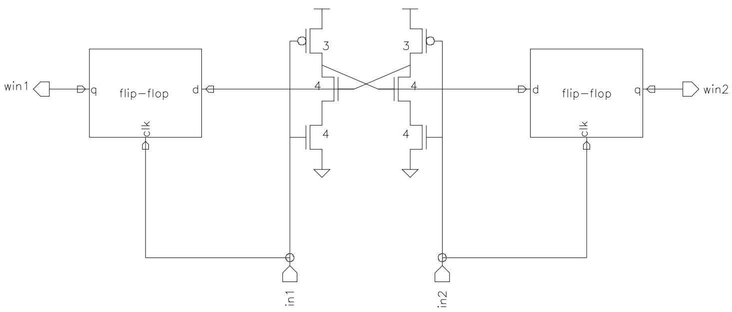

Figure 1-10: Arbiter used in the delay-locked loop

1.3.2 Instruction Distribution

A spatially distributed SIMD system must ensure not only that all clocks are synchronized, but also that all chips are executing the same instruction at the same time. Several effects complicate this requirement.

First, signal propagation time across a large circuit board is significant at high frequencies. Propagation time is approximately 2 nanoseconds per foot. Thus for a 60 cm board, chips near the distribution point receive their signals four nanoseconds before chips on the far edge. Even wires connecting to the same chip can deliver signals at slightly different times due to routing differences. For example, wires routed on outside PCB layers experience a different effective e_0 and therefore different propagation velocities. Finally, PCB thickness variations can cause impedance variations that affect signal velocity. Two common techniques to overcome this problem are snaking traces to equalize propagation delay, or by using specialized clock distribution parts with programmable delays [**?**]. Neither technique is feasible for a board with four 30-bit instruction busses running at 250 MHz.

Another source of instruction skew is the variability of discrete parts in the distribution path. For example, the clock to output delay on a high-performance ECL shift register has an uncertainty of 0.4 ns. A signal traversing two parts can be skewed by a much as 0.8 ns. The combined effect of these skews is that some chips receive an instruction a cycle later than others, even though all chips are supposed to be executing the same instruction at once. Worse yet, bits of an instruction can be mixed with bits of the next instruction due to parts variability.

Thus, the instruction distribution scheme must solve three problems. First, it must sample safely within the 4 nanosecond data window. Second, it must align all bits of an instruction to arrive at the same time. Third, it must align instructions across all chips in the system. The Abacus-1 instruction pads incorporate retiming logic, shown in Figure **??** that solves each of these problems in turn. Retiming occurs as part of the system startup process and proceeds in three stages.

The sampling alignment stage first finely adjusts the delay of each bit until the sampling clock transitions safely after the data is stable. The variable delay is performed by an all-digital delay line consisting of inverter chains and multiplexers. A binary value generated by a free-running counter selects the delay. The counter runs until the sampling clock clock transitions after a delayed version D' of that data. That event disables the counter, freezing the delay. Ideally D' is one quarter of the cycle time, so that the sampling occurs in the middle of the data valid period, but the quarter-delay line is limited to approximately 30 inverter delays. Notice that the instruction generator must ensure that the data transitions on every cycle during this phase.

Once this phase is complete, the bit alignment stage lines up all instruction bits. This technique proceeds by passing the received data bits through a shift register. The shift register outputs are ORred together to generate a `got_one` bit. As soon as all instruction pads assert this bit, the shift register is disabled. The position of the leading one selects the delay to be applied to the data of that particular pad by selecting the tap location from

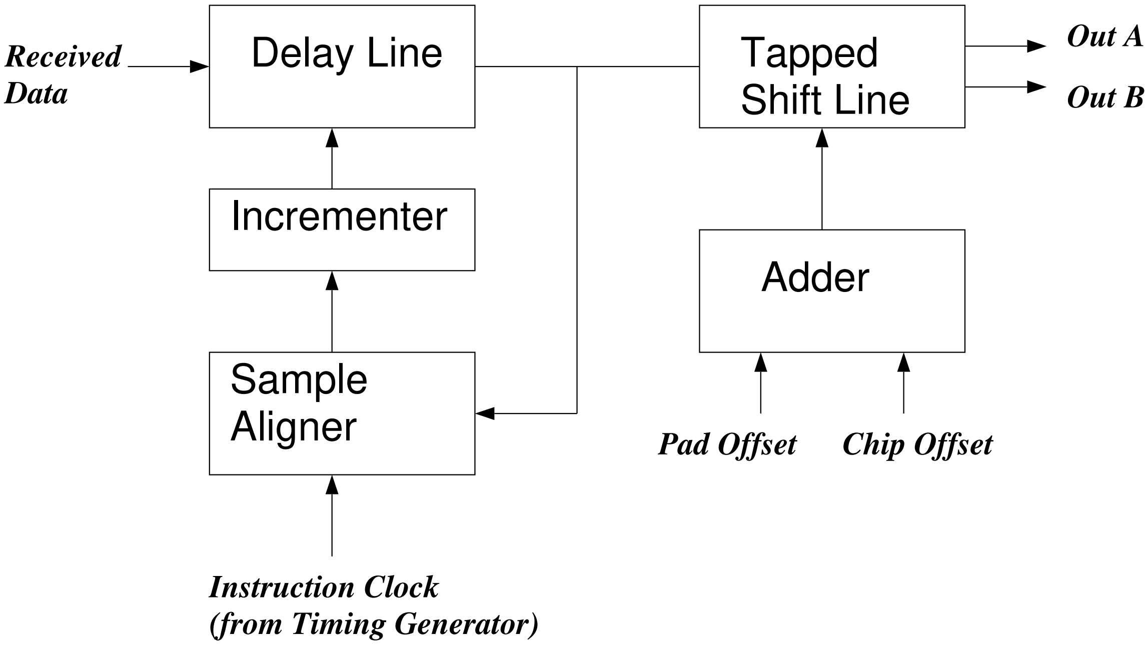

Figure 1-11: Instruction Pad Block Diagram.

another tapped delay line.

Finally, the instruction sequencer performs the system-wide alignment phase by broadcasting instructions and using the global OR feedback to determine the delay to each chip. Once this delay is known, the sequencer uses the boundary-scan interface to set a register in each chip that determines another delay to be applied to all pads in the chip. To eliminate the need for yet another delay line, the control register value is added (in unary) to the delay computed in the previous phase, and the sum is actually used to select the tap location.

The development history of this design is somewhat interesting. When it became apparent that instruction distribution time was comparable than cycle time, a proposed design used the PE chips as pipeline registers, receiving and sending instructions point to point. To synchronize instruction execution, successive chips in a column would have to execute out of different stages in an internal instruction buffer. A natural next step is unpipeline the process, and have the chips load the internal delay buffer direction from the bus. Once the idea of cycle-level delay is accepted, it leads to using nanosecond-level delays at the pin level.

1.3.3 Inter-Chip Mesh Communication

The next system level challenge is to provide synchronous high-bandwidth, low-latency mesh communication between PE chips. Due to pin limitations, the edges of the 32 by 32

PE mesh on each chip are time multiplexed onto 16-bit ports. The required clock rate is therefore 250 MHz, which presents a number of design problems.

Chips are separated by significantly varying transmission times. For example, chips on board edges communicate over wires as long as 2 feet, while neighboring on-board chips are separated by less than two inches. Also, because different signal paths have different propagation speeds. For example, the ribbon cables used between board have a characteristic impedance of 90 ohms, and propagation velocities of XX, while PCB traces have Z_0 of 50 ohms, and velocities of YY.

Process skews between chips introduce more differences. Although the chips are synchronized at the start of a cycle, subsequent edges diverge in time. Therefore, the faster chips can sample second transmitted bit too early, getting a copy of the first bit.

Unlike the instruction pads, the grid port design assumes that all wires in a port are bundled and experience similar delays. Each of the four mesh ports sends out an escort clock along with the data. Since the off-chip drivers and propagation time are identical for data and clock, the receiving chip can safely use the escort clock (delayed by the setup time of the receiving registers) to sample the data. Metastability problems do not arise as we ensure that data is sampled well before the receiving data needs to use it.

The communication pads are driven with custom 1V, on-chip series terminated drivers and matching receivers in order to minimize power dissipation and transmission line reflections [**?**]. Impedance calibration for the series termination is performed at system initialization. Each of the four ports on a chip is calibrated independently, since on-board and board-to-board impedances may differ.

In addition to the pipelined, high bandwidth mode of operation, the mesh pads can operate in a flow-through mode that bypasses the on-chip pulldown network with a single wire, allowing fast multi-chip broadcast algorithms. Propagation through a chip is reduced from three clock cycles to half a cycle. However, the time multiplexing is disabled, reducing the effective bandwidth. This was added to investigate the practicality of algorithms designed under the architectural model of meshes with reconfigurable busses.

Crossing a chip boundary costs an extra cycle for move operations. That occurs because on-chip nearest neighbor is on the edge of making it to the adjacent processor, never mind across a board. (Note that with aggressive cycle times, worst-case travel in a real system is at least a cycle time).

The edge interface does not present a uniform mesh model to the programmer.

1.3.4 External Memory Interface

Practical systems demand this. Otherwise, arch is limited to systolic algorithms, and images that fit on-chip. A true computer should be able to process very very large data sets, albeit slowly.

Since many algorithms will exhaust the 64 Kbit on-chip memory, each PE chip is backed by two 1M x 32 specialized DRAM modules. The data interface is 64 bits wide and operates at 16 ns, which matches the 32-bit, 8 ns PE mesh. The 16 ns cycle time is made possible by the Ramtron extended DRAM part, which contains an SRAM cache integrated in the DRAM.

Control signals for the memory chips are synchronized to the PE chips by sending them as part of the instruction stream. Fine control over DRAM timing is made possible by the small instruction cycle time, which is comparable to that of dedicated DRAM controller. A useful side-effect of this approach is the elimination of memory glue logic. A counter in each PE chip generates the external memory addresses. This counter can be loaded from the top row of the PE array, allowing indirect addressing.

Crucial to useful system. Considered other technologies such as Rambus and Synchronous DRAM.

Allows different memory chips by changing control signals in software.

More banks can be added with extra S0, S1 enable signals.

- Designed for block transfer, which fits well with the cached nature of the RDRAMs. The long DRAM read latency is encountered only once, after which all reads operate at SRAM speeds.
- Each RDRAM module incorporates a whole lot of cache, so actually several parallel variables are read in at once, similarly to cache lines.
- It would be nicer to use very wide, shallow chips instead of the current narrow deep ones.

1.3.5 Image I/O Interface

The next external interface requires getting image data (or any other data set) into the array. Fortunately, computer speeds have increased far faster than the bandwidth demands of images.

While early SIMD designs were concerned with limited image I/O bandwidth, it is not an issue today's high-speed designs. Since I/O is distributed among 256 chips, a single pin on each component results in a very wide bus.

The Abacus PE chip has a one-bit single-ended ECL input and a one-bit differential ECL output for external data I/O. The aggregate bandwidth of a 32-byte, 125 MHz bus is 4 GB/sec, which is almost three orders of magnitude than the 7.7 MB/sec required by 512 by 512 images at 30 Hz. The I/O bottleneck, if anywhere, is clearly going to occur upstream of the PE array. This very high I/O bandwidth makes the architecture well-suited to applications requiring real-time processing of large amounts of data such as video and synthetic aperture radar.

The I/O bandwidth can be scaled down to reduce the interface logic and wiring cost. For example, if only the 16 chips comprising the top row of the 256-chip array were connected, the 4 GB/sec bandwidth would be reduced to 250 MB/sec, which is more than an order of magnitude higher than frame rate.

A single PE instruction initiates a burst transfer of 32 bits from the edge of the PE mesh to the output pin (or vice versa). This background transfer frees the array to perform computation, or in the case of the lower cost interface, to use the mesh connections to shift data towards the connected chips.

A single chip has 1.5 MB/sec bandwidth. Of course, it must store this data somewhere, which it can do, if it does no further processing. Transfer path to memory is 40 cycles for a 1Kbit write, which is 3.2 bytes/cycle,w which is 4GB/sec. So the memory bandwidth can easily keep up with IO bandwidth.

1.4 Chip Organization and Control Logic

The chip is organized into three major parts: the PE array, split into two halves; the control signal drivers in the middle; and control logic at the left. The control logic includes address decoders, instruction decode, external memory control, and a scan path controller. The PE core is organized as 16 strips of 64 PEs each. Each strip also contains two sets of 76 buffers that drive control signals outward from the center. Skew is reduced by distributing signals vertically in a lightly-loaded metal-3 bus, buffering horizontally with identical drivers, and equalizing the load on each horizontal control wire.

The Abacus PE chip contains several mechanisms required for high-speed system-level operation: skew-tolerant instruction distribution pads, high-bandwidth local DRAM interface, low voltage swing impedance-matched mesh pads, and low pin-count I/O interface.

System	Area mm^2	Fraction %
Pad Ring	24.23	25.4
Control Logic	9.43	9.9
PEs	38.30	40.1
Row buffers	7.95	8.3
Busses	13.09	13.7
Edge Interfaces	2.57	2.7
Total	**95.57**	

Table 1.4: Chip Area by Subsystem

The PE array formed approximately 42% of the chip area and 56% of the chip core area.

1.4.1 Timing Generator

The timing generator is a straightforward design, based on tapping a delay line composed of inverters. Clocks are generated by ANDing a tapped signal and a delayed version. This

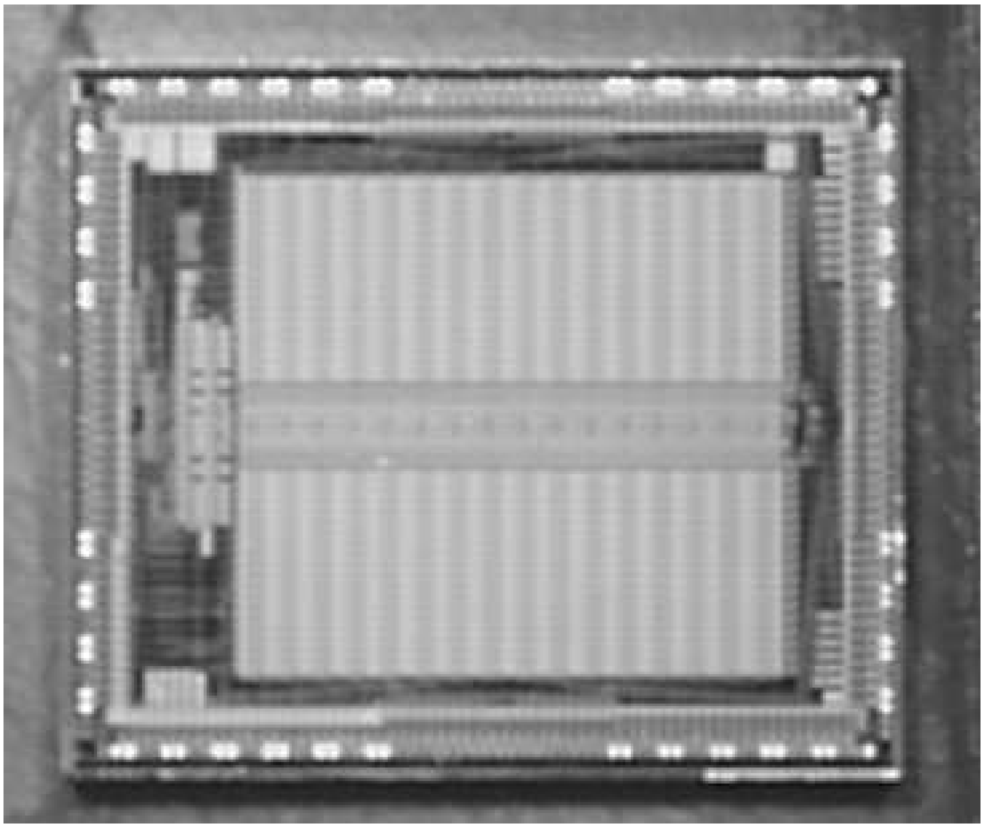

Figure 1-12: Chip photograph.

method provides an accurate method of generating pulses quantized in terms of inverter delays. Process variation is tracked nicely. Another nice feature is that all clocks are generated from the leading edge of the delay line input.

The timing generator includes a debugging feature that doubles the delay of each inverter stage by opening a pass gate to a capacitor. This technique stretches out the clock waveforms uniformly, affecting both pulse widths and pulse start times. The feature was intended to allow testing of the chip in case the original pulse widths were too narrow. The original design was based on an analog-controlled delay line [?], where the voltage controlled the gate voltage on the switch. This flexibility was unnecessary and routing an analog line to each chip in a system was judged to be impractical.

1.4.2 Pad Ring Summary

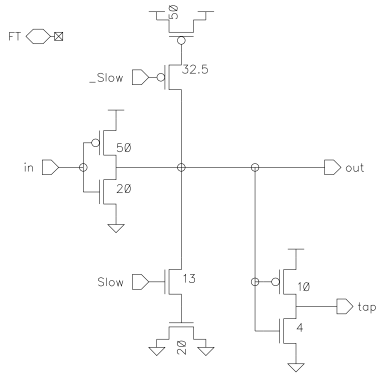

Figure 1-13: Time Unit.

Pins	Number	Direction	Voltage	Freq	Load
Grid clock in	4	I	1V	250	NA
Grid measurement	4	O	1V	250	NA
Grid clock out	4	O	1V	250	1V Pad
Grid data	64	IO	1V	250	1V Pad
ECL Vref	1	I	ECL	DC	NA
Clock	2	I	ECL	125	NA
Instruction	30	I	ECL	250	NA
DRAM address	11	O	CMOS	62.5	73-100 pF
DRAM control	7	O	CMOS	62.5	28-101 pF
DRAM data	64	IO	CMOS	62.5	8 pF
IO Port data	3	IO	ECL	125	ECL SSI
Global OR	2	O	ECL	125	ECL SSI
Boundary scan	5	IO	CMOS	125	NA
Debug	4	O	CMOS	125	10 pF
Total signal	205				

Table 1.5: Pin Requirements.

1.4.3 Instruction Decode

Instruction decode is very simple, mostly because the instruction word is effectively horizontal microcode. The decoder consists of an equivalent of approximately forty two-input gates. A sizeable fraction of those gates detect when a special instruction is specified. Once that occurs, further decoding of special instructions requires at most two more gate delays. Ordinary instructions are not decoded at all; addresses are sent directly to the address decoders and ALU op codes directly to the pipeline registers. Including the address decoders, the logic depth of the decode stage is approximately eight gates.

1.4.4 TAP Control

The Abacus chip incorporates a standard IEEE Test Access Port (TAP) interface. This interface is used not only for testing the chip, but is a necessary part of system initialization and configuration. It is used to initiate clock synchronization and instruction pad retiming, to program impedance settings on the grid communication ports, and to inform each chip about its location on the circuit board.

Testing functionality of the interface includes access to the decoded instruction pipeline registers. These registers can be scanned out to evaluate instruction pad functionality or scanned in to override the instruction decoder and allow testing of the PE array in case of a design fault.

The current design has a fault in that the initial value of the pipeline registers upon powering up is unknown. As a result, multiple word lines can be activated.

1.5 Design Style

The high clock speed of the Abacus-1 chip is due to careful circuit design and physical layout. This section identifies the factors that contributed to fast execution.

Layout Style. The processing element was implemented entirely in full custom using the Cadence tool suite. A number of small standard cells used in the control logic were synthesized from schematics. The control section, including instruction decoding and TAP control, was assembled from these cells and interconnected with a greedy router locally developed by Larry Dennison. The router introduced some layout violations, and those errors were manually corrected.

Layout synthesis was also used in creating the scan registers surrounding the instruction decode, but this turned out to be a mistake. A rough estimate indicates that a manual layout would have required approximately the same amount of design effort, and would have achieved almost 30% area reduction.

A substantial contributor to layout density was the use of vendor-specific layout rules instead of the common scalable design rules provided to universities by MOSIS. The effort invested in customizing the Cadence tools to the Hewlett-Packard technology was quickly repaid, as the area penalty due to scalable rules is estimated at 40%, especially for full custom layout.

Dense layout decreases wire length and parasitic capacitance.

Circuit Design. A number of design principles independent of layout optimizations contributed to the high clock speed.

1. Dynamic logic was used extensively in the ALUs, network configuration decoder, and the network pathway itself.

2. As a corollary, the design was optimized to use mostly N-fets for pass gates and precharge transistors. Not only are N devices smaller, leading to denser layout, but they also load the control signal much lighter.

3. Global signals were carefully distributed by the use of matched wiring, driver buffers, and dummy logic elements. As a result, timing predictability was higher, and smaller timing margins could be used.

4. Some circuit elements were used for multiple purposes. For example, precharging bitlines through the column select transistors required only one precharge transistor per bus, instead of one per bitline.

5. Noise margins were sacrificed where necessary. For example, the bitlines were prechared to a V_T drop below V_{dd}. Also, inverters were ratioed based on preferred transistion directions. Thus, if a particular node was precharged high and evaluated low, the corresponding inverter was ratioed to switch a a higher voltage.

6. Dynamic latches were used to capture data and allow the previous stage to be reset as soon as the data was latched. This increased the effecive recovery time.

7. Buffer size and scaleup ratio was optmized in both layout and in matching the load. A tool was written to probe a wire in a schematic and calculate the total effective loading on it. As the design changed, the program constantly recalculated optimal buffer sizes. Buffer layout tool was optimized to generate a fingers-style layout, greatly reducing parasitic diffusion capacitance.

Functional Optimizations. Some performance optimizations relied on the use of software to compensate for layout quirks. For example, reading from the network port produced the opposite of what was written. Correct execution depended on the assembler to detect the network port as one of the sources and modify the ALU operation, relying on the flexibility of the ALU to generate a compensating function.

Another class of optimizations arose as a result of removing abstraction boundaries. For example, reading from the $\overline{\text{bit}}$ bus results in inverted values, but this value need not be corrected until it reaches the ALU inputs, and there the correction need not add an inverter delay but simply reverse the outputs of a local buffer.

1.6 Testing the Chip

This section describes the methodology and results of testing the chip. Its primary purpose is to document the testing environment in order to validate the test results measurements.

The test setup consisted of four hardware and software modules: the test board itself, a clock driver, a scan controller, and a Sun workstation. The test chip was mounted on the test board described earlier. The clock was provided by a clock board which was essentially an ECL flip flop in a divide-by-two configuration, connected to an oscillator socket. The socket was driven by an ECL-level oscillator or a waveform generator for high and low-speed tests, respectively. The flip-flop ensured that the clock duty cycle would be close to 50%. The scan controller was a DSP board based on the TI C30 processor. The controller interfaced with the test board through a four-wire CMOS- level IEEE TAP interface. A Sun workstation communicated with the controller through a conventional RS-232 interface.

1.6.1 Test Board

The test board consisted of 12 layers, as described in Table **??**. As well as testing the chip, designing the board served to demonstrate that routing of the fairly wide busses, both instruction and grid, was possible. Six signal layers proved sufficient to route all signals. The test board also demonstrated that PCB design software was able to route differential ECL signals well.

The board wiring methodology was controlled impedance *offset stripline*, with horizontal and vertical pairs of routing between two power or ground planes. The planes isolated the layers, and the orthogonal orientation between the planes ensured that signals within the same ground plane pair did not couple over long distances.

The board accepted four separate power supplies. Even though the ECL circuits were operated at CMOS levels in offset mode, between 0 and 5 volts, instead of -5.2 V and 0 V, the board also included a termination voltage plane, and a 1 V grid signalling plane.

Layer	Name	Orientation	Purpose
1	TOP	Random	SMT/via connectors.
2	V_{TT}	Plane	ECL termination voltage. Tied to 2V.
3	X1	Horizontal	Routing
4	Y1	Vertical	Routing
5	V_{CC}	Plane	Positive power supply. Tied to 5V.
6	X2	Horizontal	Routing
7	Y2	Vertical	Routing
8	V_{EE}	Plane	Negative power supply. Tied to 0V.
9	X3	Horizontal	Routing
10	Y3	Vertical	Routing
11	V_{CE}	Plane	1 V power supply. Tied to 1V.
12	BOT	Plane	Negative power supply. Tied to 0V.

Table 1.6: Board Stackup

The top and bottom of the PC board were not used for wiring. The bottom was unused because the dense wiring area around the PE chip was covered by the interposer clamp block.

1.6.2 Test Process

The chip was tested through the Test Access Port (TAP) interface. The test flow was thus:

1. The user writes a C-based test program on the Sun workstation using calls to the Abacus assembler library.
2. The test program runs on the Sparc and generates a driver C program targeted for the DSP board.
3. The user downloads the driver object code to the DSP board.
4. The driver runs on the DSP board and uses the C30 digital I/O ports to control the TAP interface on the Abacus chip.

The clock board was driven by one of two crystals, either 200 MHz or 160 MHz, or a waveform generator with a maximum frequency of approximately 25 MHz. Since the clock board divided the clock by two, the resulting chip test speed was either 100, 80, or 12.5 MHz.

1.6.3 Test Results

All data shown in this section was obtained with a Tektronix TDS 460 digital sampling oscilloscope. The probes were custom low-capacitance 21:1 shop probes made with a 1 KΩ isolating resistor and RT58 coax cable.

Heartbeat monitor. The very first test was to ensure that the differential ECL clock signal was received properly and reached the on-chip timing generator. One of the four debug pins was a copy of the `Read` clock taken from the eastern edge of the PE array. This 3.3 ns pulse is shown in Figure **??**. As expected, the pulse occurs a fixed amount of time after the rising clock edge and is not affected by the duration of the clock period.

TAP Interface. The next testing step was to communicate with the TAP interface. Figure **??** shows the output of the serial *TDO* (TAP Data Out) pin after the reset signal was asserted. This test demonstrated that the TAP was functional and that further testing could proceed.

IO Burst. A bit pattern was loaded into the top PE row with the literal load command. It was then transferred to the IO plane, and then to the output register. Finally, the IO burst was enabled to produce the waveform shown in Figure **??**. The specific bit pattern is 0xFF504A. The pattern is reversed, since the LSB appears first on the output of the scope. The ground/supply voltage bounce is about 200 mV. Although only a single- ended trace is shown in the figure, the output is driven differentially, eliminating the common-mode bounce.

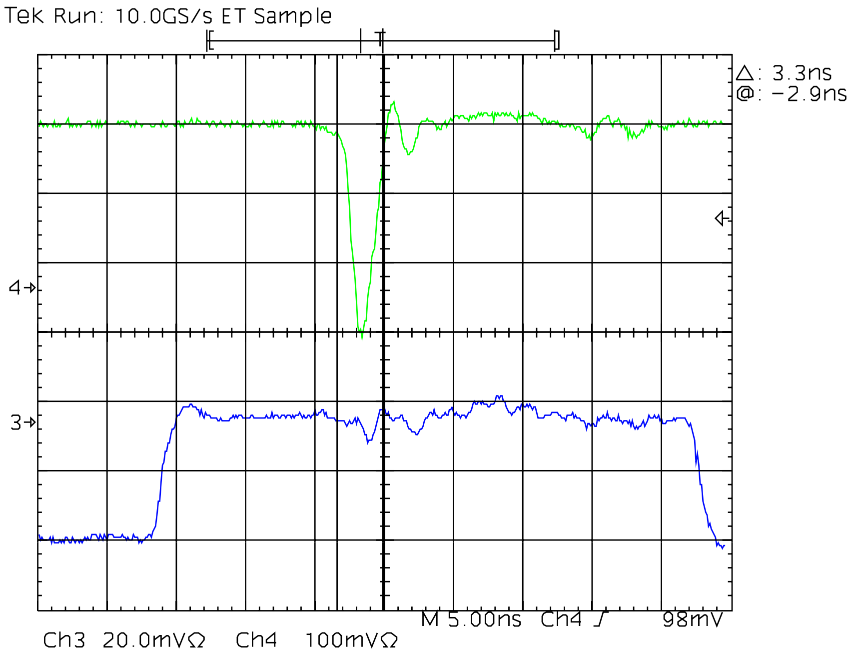

Figure 1-14: The heartbeat monitor: a copy of the read clock.

Data Plane Shift. The next test involved testing the operation of the data plane. The DP register was set high, and then the data plane shifting was enabled. The DRAM I/O pins floated low, so that a zero propagated through the data plane. As expected, after thirty-two cycles, the zero appeared at the top row. For this test, the global OR output pin was used, testing both subsystems.

External Memory Pads. The DRAM pads were toggled at 100 MHz.

Register File Toggle. The next test showed the first failure of the chip. The repeated instruction was very simple: toggle a memory bit and copy it onto the burst output pin. This operation started failing at 80 MHz. Instead of a pattern of alternating zeroes and ones, the pin showed two high values followed by a low.

Grid Shift Operation. The on-chip grid communication was tested next. As shown in Figure **??**, the pattern was shifted out. Although only 20 MHz operation is shown, the grid shifting worked at 80 MHz, and showed signs of failure at 100 MHz.

1.6.4 Testing Summary

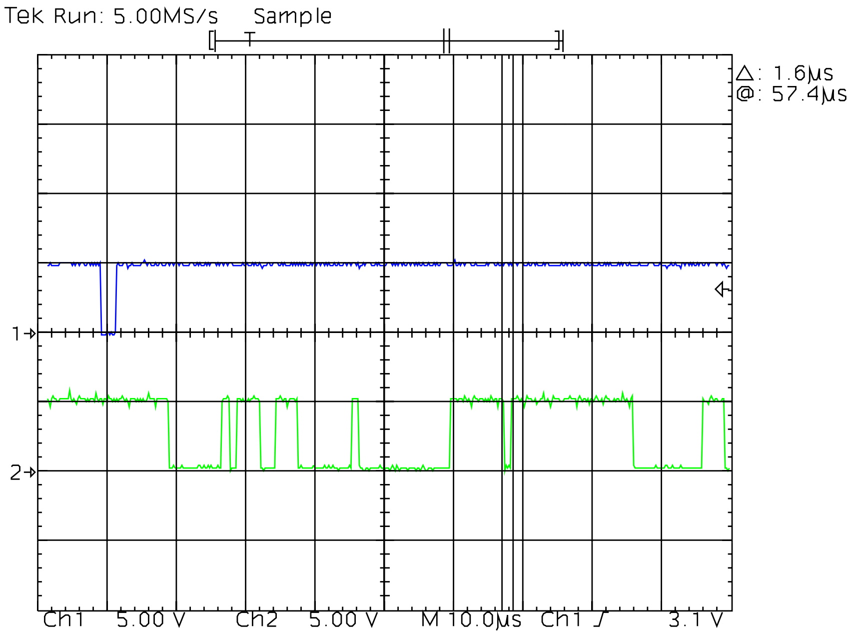

Figure 1-15: TAP Interface

Subsystem	Status
Burst Output	Operational at 100 MHz
Internal Memory toggle	Operational at 80 MHz
On-chip grid communication	Operational at 80 MHz
Data Plane shift	Operational at 100 MHz
TAP interface	Fully operational

Table 1.7: Testing Summary

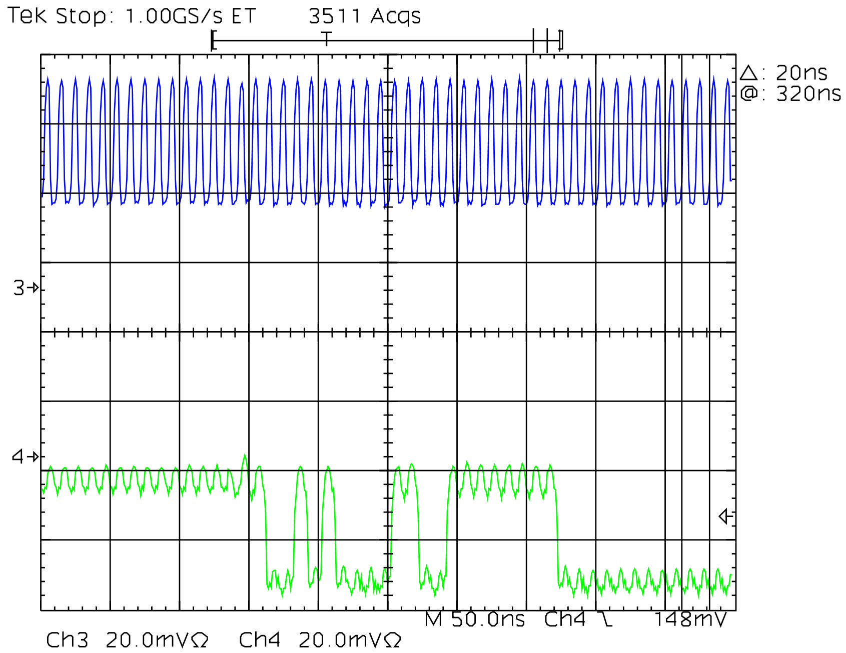

Figure 1-16: Output Burst Trace, 100 MHz.

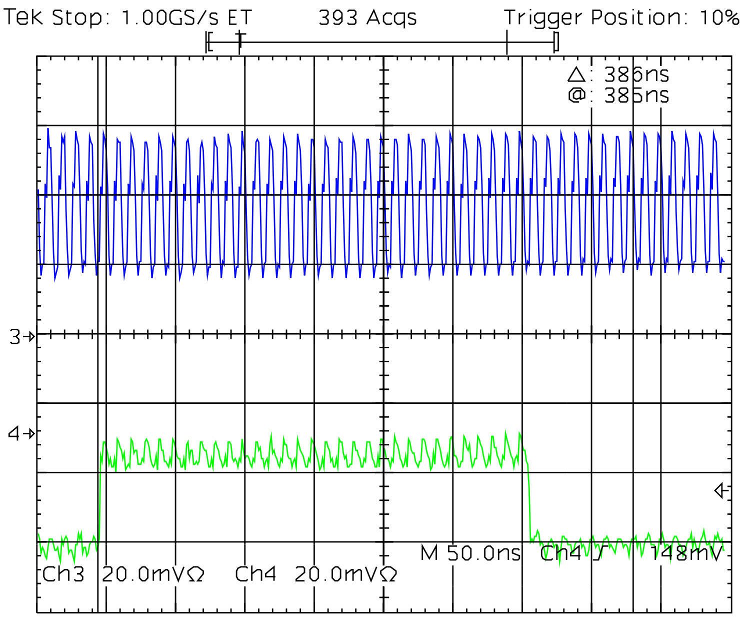

Figure 1-17: Data plane toggle at 100 MHz. The data plane is loaded with 1s. A zero is then introduced at the input. 32 cycles later the 0 is has been shifted through.

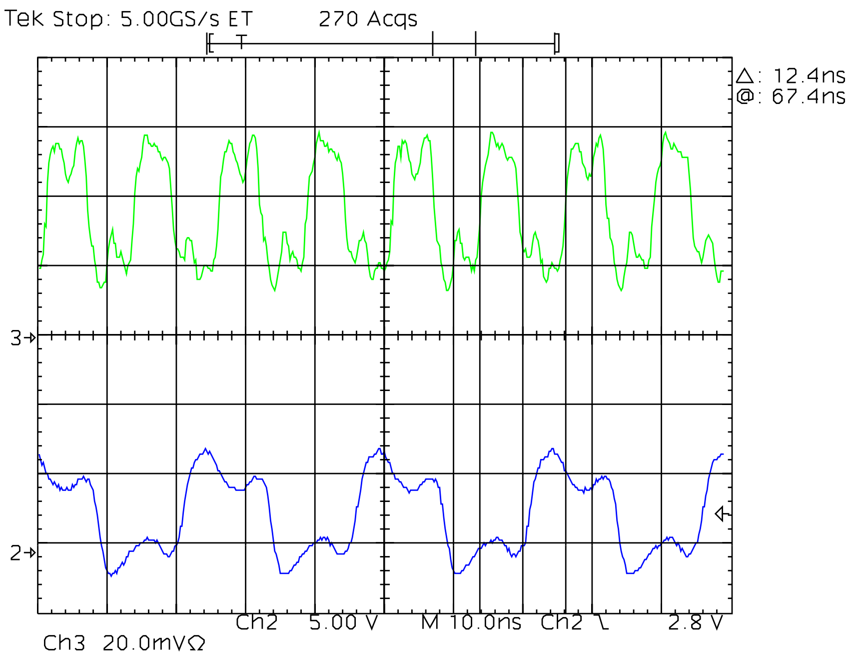

Figure 1-18: DRAM pads were designed to toggle at only 62.5 MHz.

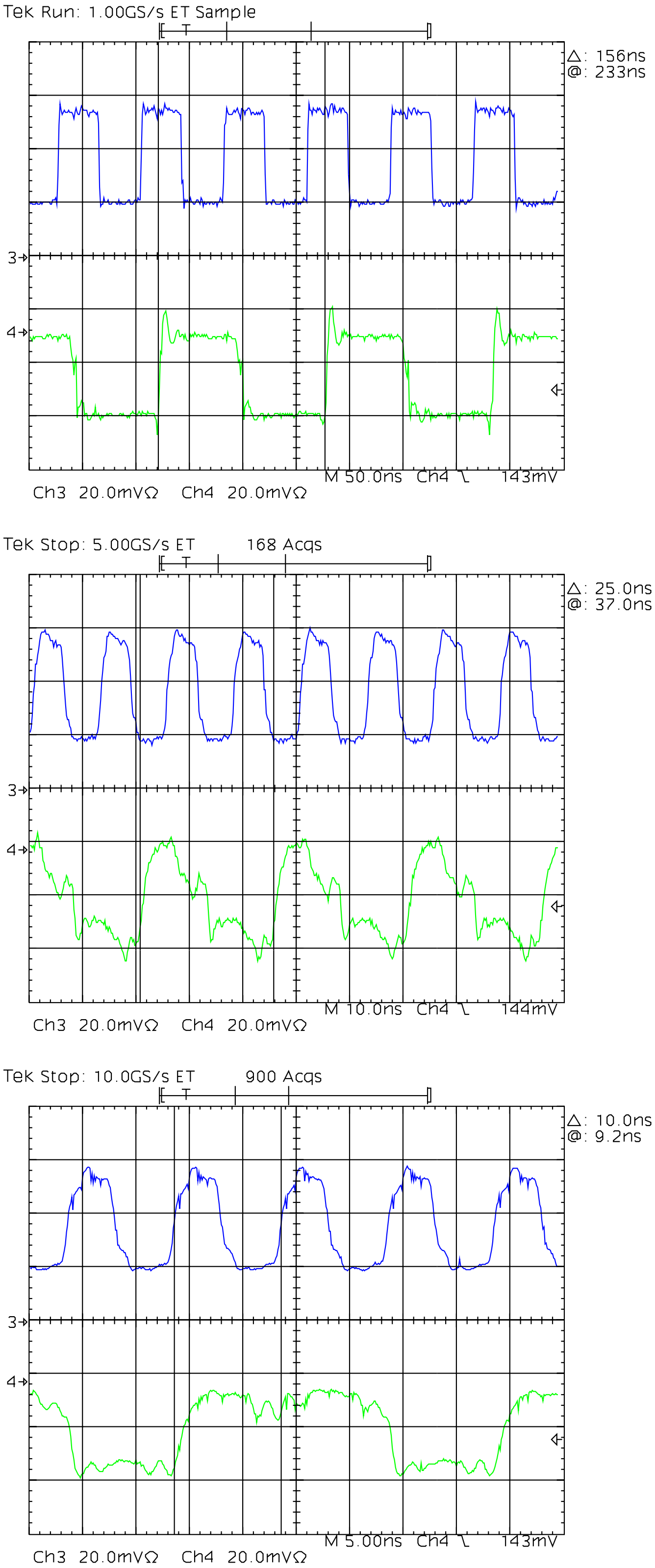

Figure 1-19: IO Toggle

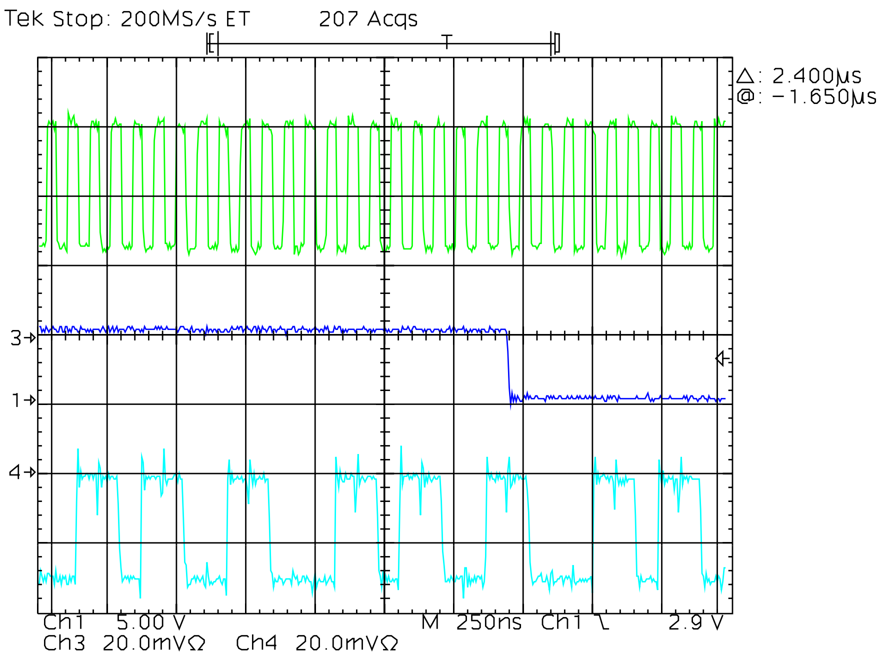

Figure 1-20: Grid shifting at clock rate. Only the 20 MHz operation is shown.