Chapter 1

The Next Generation

After designing the Abacus-1 chip, a number of design flaws and potential improvements became apparent. In a sense, the discussion belongs in a conclusions chapter, but it is a prerequisite for the design of the next generation.

The goal of the design changed, from building a supercomputer-class machine to a co-processor device for a workstation. Most significantly, having shown that high speed design is possible, we encountered power dissipation as the main bottleneck.

This chapter incorporates the lessons of Abacus-1 into the design of the next generation machine, Abacus-2. Although presented in the context of a low-power, low-cost coprocessor, the design can be converted back to a supercomputer-style system by returning to large, fast chips housed in specialized packages and cooled by carefully designed fan systems.

The implications of a co-processor style design are:

- Low power.
- Low cost packaging.
- High virtualization factors.
- More off-chip memory per PE chip.

- No specialized IO boards.

Our preliminary design for the Abacus-2 part addresses these issues, among others. The design retains constant performance, while halving the chip area, reducing power dissipation, and decreasing the required packaging cost. The Abacus-2 chip fits 64 8-bit PEs on a single 8 x 8 mm die, and operate at 66 MHz in a 3.3 V (or lower) environment.

A single Abacus-2 chip will deliver 4.2 billion 16-bit additions per second and approximately 500 million 8-bit multiplies per second, in a non-aggressive 1 micron VLSI technology. Implementation in the sub-micron technology recently available would of course lead to higher performance numbers.

The new design is amenable to integration with a simple sequencer and I/O subsystem, allowing scalability in system size, from a 16-chip 64 GigaOps system that can be utilized as a co-processor attached to a conventional RISC-chip based node of a MIMD system, to a standalone 256-chip TeraOp SIMD machine. The Abacus-2 chip is designed to operate as smart memory, accessible by the controller with low latency, and without intervening corner turning circuitry.

1.1 Lessons of Abacus-1

1. *Reconfiguration overhead is excessive.*

   The design of the on-chip interconnection network requires a reconfiguration cycle whenever direction of data flow changes. This occurs when, for example, an ASR (arithmetic shift right) instruction, in which bits flow from left to right is followed by an ADD instruction, in which the carry bit flows from right to left. Although the cost is only one cycle, most arithmetic operations or move require only three or four cycles, so that reconfiguration costs approximately 20% of performance.

   This problem was not obvious at first because initial program coding was done manually, and an experienced programmer could reorganize the code to group together instructions of a given orientation, reducing reconfiguration overhead by approximately factor of two. Only when the compiler became operational, and performance dropped unexpectedly, did the flaw become obvious. Due to time constraints, there was no opportunity to investigate compilation strategies for automatic reorganization. Instead, we added a peephole optimization step that noted when two successive instructions required identical configuration, and used a non-reconfiguring version of the microcode for the second instruction. Neither software technique helps in the very common case of rapidly alternating mesh moves and arithmetic operations.

2. *Arbitrary data formats are expensive.* The one-bit bit slice elements could be grouped in almost arbitrary configurations, as long as each set of PEs responsible for a data word formed a topological circle. This choice required the overhead of network and configuration circuitry at every bit of a datapath. Additionally, on-chip data could not be accessed by the sequencer without a complex corner-turning reformatting process that requires dedicated hardware.

   In retrospect, this flexibilty is mostly useless. The constraints on cluster organization is that they must be tile-able, which essentially means rectangular, and both dimensions must be a factor of 32 to fit evenly onto a chip. These constraints allow a limited number of configurations, so the bit slices may as well have been grouped more efficiently to begin with.

3. *Rise/fall times are a substantial fraction of cycle time.* When rise/fall times approach 400 picoseconds, three or four non-overlapping control signal transitions consume as much as 3 ns. When combined with safety margins for process variation uncertainties, the actual active time shrinks further. For example, an address wire must transition four times during a cycle: one each for the read and write phases, and once for the precharge phases between the read and write. The 1.6 ns of rise/fall time is fully 20% of the cycle time.

4. *Simple ALUs do insufficient work per cycle.* The very simple ALU operates in approximately 400 picoseconds, a tiny fraction of the overall cycle time. The common

battlecry of "keeping the silicon busy" does not completely apply when computation actually occurs during 5% of the cycle time. This is a central weakness of conventional bit-serial SIMD systems: they spend far more time fetching and storing data than actually operating on it. The implication of this analysis is that the main constraint on ALU complexity is area, and not computation time.

5. *Pipelining is important.* The Abacus-1 design could not be effectively pipelined since much of its bit steering depended on network reconfiguration, which was manipulated by writing control registers. Forwarding could not be used because the arriving bit was a function of (as of yet unwritten) control registers in many other PEs. As a result, the very common network operations would have introduced pipeline bubbles.

   Without pipelining, a single cycle had to encompass the read, write, and execute phases. Cycle times thus stretched considerably longer than necessary.

6. *Double-banked register files are expensive.* Although the double-banked register files and twin ALUs allow two result bits to be written per cycle, the cost of this flexibility is higher than first expected. As discussed in the Chip chapter, the PE area impact is not significant. However, the required board-level instruction bandwidth is almost doubled. An intermediate possibility is to use double banked files but generate adresses from the instruction code. For example, the carry and sum bits could be written to identical addresses in the two banks.

7. *ECL components lead to low integration.* ECL signalling was chosen for instruction distribution and I/O paths before the system design was complete. The high parts count of SSI ECL components did not become apparent until after the chip was designed. So although the interface to the external memory chips is glueless, each PE board has two SSI chips per PE chip. In addition sheer number of chips, ECL designs are complicated because they use different voltage levels, discrete terminators and bypass capacitors, and a more complex circuit board structure.

   As pointed out in Section 666, the discrete circuitry could be eliminated with two custom ASICs. But the question arises: if a special chip is required, why not use

CMOS signalling to begin with? For example, at lower frequencies, low-swing CMOS signalling consumes significantly less power than that dissipated in ECL terminators.

8. *Burst I/O is overconstraining.* The Abacus chip allocates three pins for data IO. A single I/O instruction initiates a 32-bit burst at 125 MHz. This design choice was intended to both increase the overlap of computation and I/O by requiring only a single instruction per 32 bits, and to reduce the number of pins on the chip.

   The design also places heavy demands on system-level IO reformatting circuitry. It must capture 32 256-bit words arriving at 8 ns intervals. High performance parts are required to handle this quantity of data, yet they sit idle almost all of the time, as I/O transfers are very rare.

9. *Network model must be maintained transparently between chips* Abacus-1 implemented a reconfigurable mesh network. PEs could be linked together by almost arbitrary topologies, and operate as a multiple-writer bus. This is a powerful machine model, and is the subject of considerable algorithmic research. Unfortunately, the multiple-writer model breaks across chip boundaries, therefore could not be used for system-wide algorithms.

1.2 Abacus-2

In order to see how each of these lessons was applied to the Abacus-2 architecture, this section gives a quick overview of the design. An Abacus-2 processing element consists of an 8-bit ALU, 512 bits of single-banked memory, a carry bit interconnection network and a grid interconection network. The carry network provides two unidirectional links running in opposite directions. The grid interconnection network allows PEs to select data from each of their four neighbors.

Abacus-2 Picture here

Figure 1-1: Abacus-2 Interconnection

1.2.1 Pipelining Analysis

Processor performance is frequently improved through pipelining the various phases of the clock cycle. Pipelining involves adding registers between circuit stages to allow parts of different instructions to execute at the same time. As long as the pipeline is kept full, the throughput rate of a pipelined processor increases by a factor equal to the number of stages. Most modern microprocessors have between 5 and 15 pipeline stages.

Pipeline performance degrades when data or control hazards occur. For example, if a register is written in one instruction and referenced in the next, the computed data item has not yet been written to the register file and therefore cannot be read. The pipeline must be stalled until the data arrives. Alternatively, a forwarding path can be added between the ALU input and the write stage pipeline register, so the data can be directly shunted without a pipeline bubble. This forwarding circuitry as well as pipeline registers are the main area costs of a pipelined implementation. In conventional processors, the area cost is negligible compared to the four-fold performance gain. This advantage is not quite as clear for SIMD systems, where the entire datapath is no bigger than two or three registers.

SIMD systems encounter three other hazards that decrease the effectiveness and increase the cost of pipelines. First, communications operations occur frequently in image processing code. Every such operation introduces a stall in the pipeline, since network data, unlike internal data, cannot be forwarded.

Second, the value of the *active* register that controls conditional execution must be forwarded to the register file writeback controller, as well as the global bit that disables the writeback control so that the *active* register may be written. An alternative to expending area on forwarding circuitry is to stall the pipeline in software.

Third, unlike conventional pipelining, when the forwarding logic detects the read of a register that is still being written, it cannot simply forward the write stage inputs to the ALU. The problem is that the data on the write stage inputs may not be actually written to the register file (based on the *active* bit), and it would therefore an error to forward that data. The ALU actually needs the value stored in the register file. Thus, writing a value and reading it out immediately causes a stall, unless the compiler is certain that the active bit is off.

The benefit of pipelining in a system already operating at the top clock rate allowed by the instruction delivery system is not expressed in performance gains. Instead, power can be reduced by lowering the voltage and therefore slowing operation back to the unpipelined rate. Since power dissipation is the next barrier to high performance, this tradeoff appears worthwhile.

A standard three-stage pipeline consists of read, execute, and write phases. Two pipeline registers and a three-to-one multiplexer are required for each phase per data port. Assuming dynamic circuits, the area cost is estimated as approximately 30 inverters per datapath bit. This is approximately 50% of the existing non-memory circuitry.

A more significant area cost is the growth of the memory cell due to the addition of an extra write port. A typical area ratio between a two and a three-port cell is approximately 1.5. The combined ALU and memory-caused area increase is thus also 1.5.

The increased throughput thus comes at an area cost of approximately 60%, and an additional latency of two gate delays in each cycle. Assuming the clock rate can be increased (or voltage lowered), the overall performance improvement is a factor of approximately 1.8.

1.2.2 Fixes of Abacus-1

Reduced Reconfiguration. Reconfiguration overhead has been almost completely eliminated in this design by two major changes. First, reconfiguration within arithmetic operations has been reduced by providing two wires for bidirectional signalling between bit slices so that computations flowing in opposing directions (such as an add followed by a shift right) do not require reconfigurations.

Second, reconfiguration has been eliminated for inter-pixel operations by separating the roles of the carry chain and the mesh communication. The design recognizes that for mesh operations, all pixels usually communicate in SIMD mode, and do not use the "current neighbor" facility of network configuration. Therefore, the direction of mesh inputs is selected globally for all PEs.

This redesign allows reconfiguration to be eliminated completely except for changes in the word unit (WU) size. As a result, the configuration register write time can be very slow. The current design does not take advantage of this optimization opportunity.

Fixed Data Formats. The word format ordering has been fixed so that bits are oriented horizontally, with the LSB on the right side. This addresses the issues of constant distribution and corner turning. The implementation allows simple, efficient constant generation by the sequencer with only a few byte shift instructions. Similarly, the four typical word formats (8, 16, and 32) can be easily manipulated even in a bit-serial stream. The disadvantage of this organization is that bits are no longer configured in a nearest-neighbor head-to-tail snake pattern, and therefdore propagation times between slices are longer.

Rise/Fall Time Issues. These problems have decreased in importance with the lengthening of the read and write times due to pipelining.

Double Banked Register File The Abacus-2 design uses only one memory bank. The instruction word size has therefore been reduced by three addresses and an ALU operation, while each of the remaining three addresses has grown by one bit, for a total reduction of 23 bits.

Low Integration of ECL Components The instruction delivery interface is redesigned to use CMOS circuits. A custom CMOS chip provides instruction distribution to two Abacus chip columns.

Burst IO Has been eliminated in favor of an 8-bit dedicated output port. Each column of PE chips shares an I/O bus. Each chip is given a row position identifier, and IO requests specify that position. For example, an IO instructions will specify that byte 3 of chip 1 be placed on the output bus.

An idea to be explored is the use of the wide instruction word to transmit data back to the bus master chip. In this approach, an IO instruction will cause the master to stop driving the bus, and the processor chip in a particular row to drive the contents of a PE row onto the instruction bus, while executing a NOP internally. Although peak computing power is slightly reduced, the use of a wide, 32-bit bus increases IO bandwidth to 1 transfer per 2 cycles per column, or 4 bits per chip.

Alternate Active Bit Abacus-1 supported the use of an arbitrary bit in memory as a steering bit in a multiplexing operation. A useful application of this capability is to conditionally update a value in memory by selecting between the old value and the new one. As discussed in section XX, this technique reduced operations on the active register and therefore memory traffic, especially in high VF operations.

Since Abacus-2 does not have three read ports, the multiplexing trick cannot be used. Instead, a temporary active register is conditionally ANDed with the primary active register based on a global signal.

External Memory Interface Ideally, the external memory interface is ideally suited to a high throughput, block-transfer based memory technology such as RamBus. Practically, that technology is not yet easily available. The design therefore resorts to off-chip SRAM instead of DRAM. This approach reduces the memory capacity as compared to the current DRAM, but eliminates the need to send carefully-timed DRAM control signals through the instruction stream and to design lockout circuitry to handle refresh timing.

Several suitable memory chips are available, including the 32K x 32 Micron 10 ns synchronous SRAM. With this IC, memory capacity can be expanded at the finer granularity of 1 Kword per WU. More address pins are required than for the DRAM, since the row and column addresses are no longer multiplexed. External chip select pins provide for future memory expansion.

Network Model One of the main drawbacks of the reconfigurable mesh model supported by the Abacus-1 design is its inefficient support of images larger than the physical array. Although signals propagate at electrical speeds, they must be re-registered and digitally updated at each iteration. The problem is that signals may propagate in arbitrary directions, so that virtualization reduces to cycling through all array tiles at every iteration.

Abacus-2 uses the theoretically weaker but practically more efficient model of a reconfigurable bus. In this model, each PE can select whether to output its own value, or that of its downstream neighbor onto the bus, as shown in Figure 1-2.

Speed estimation is straightforward. If bypass is implemented with a buffered multiplexer, propagation past one PE requires approximately 0.6 ns. An 8-PE row presents a delay of only 4.8 ns. Chip crossing delay is estimated at 3 ns, for a total chip delay of approximately 8 ns. The time-multiplexed mesh pins are clocked so that propagation cannot be flowthrough, and therefore incurs quantization delays. On the other hand, the clocking allows signals to be pipelined. A message can cross a chip and board boundary in one 10 ns cycle.

Virtualization is easily handled by using a tile strategy instead of a neighborhood strategy, Mesh Network

Figure 1-2: Abacus-2 Mesh Network

as described earlier. Propagation occurs at electrical speeds across a tile. The data is then registered under software controll, and the physical array simulates the next tile. Since reconfigurable busses operate in one dimension at a time, and in one direction at a time, this approach is very efficient.

Multiplies The Abacus-2 design can perform multiplications at the rate of three cycles per multiply step. Thus, an 8-bit multiply requires 24 cycles and produces a 16-bit result. The aggregate computing rate is 64 multiplies every 24 cycles, or 2.75 multiplies per cycle. Assuming a 66 MHz clock rate, this translates into 183 MOPS.

If multiplies become more important in applications, multiplication can be substantially improved with dedicated hardware. All of these approaches are more effective with a wider slice. Approaches range from:

- A dedicated shift register and Booth recoder to halve the number of iterations, and reduce the iteration time from 3 cycles to two. Estimated computation time is 4 iterations of 2 cycles each, for 8 cycles plus 2 overhead cycles. Estimated performance is therefore 10 cycles, or 6.4 multiplies per cycle, leading to 425 MOPS.
- A self-timed iterative multiplier with a Booth recoder. Estimated performance is dependent on cycle time, but a single iteration of a 16-bit add should take about four

nanoseconds. This requires four iterations, or two 10 ns cycles plus a carry computing cycle at the end. At 21 multiplies per cycle, the aggregate performance is 2.1 GOPS.

Global OR Unlike the Abacus-1 design, PEs can no longer be joined into one large electrical net. Specialized circuitry is therefore provided. The precharged carry circuitry is optionally connected to a dedicated column wire. The eight columns are reduced to a single bit by a wide conditional pseudo-NMOS NOR gate.

Scalable I/O Scalable I/O should allow the sequencer to access any pixel in the system quickly. This could be done by adding a port to the PE chip or by accessing the external memory. For small systems, the port idea works well, since the access port can be placed on a bus and tristated. For larger systems, discrete tristate parts would have to be added to isolate the bus from capacitive loads. Corner turning can be incorporated on chip, unlike Abacus-1, since the data formats have been restricted somewhat. The only complex part would be the addressing, and that could be taken care of by another set of PALs.

A Conventional Datapath The Abacus-2 datapath strongly resembles that of a conventional RISC microcontroller. Consider the 16-bit microcontroller described in a standard contemporary textbook [Weste]. Each datapath element consists of the following elements:

1. Literal multiplexer. Forces a constant onto one of the ALU ports.

2. Adder. Includes a propagate/generate unit, a Manchester carry stage, a summer, a datapath-wide zero detect circuit, and a conditional negate/zero unit.

3. Boolean unit. Performs all 16 two-input one-output data functions.

4. Shifter. A four-bit shift element.

The Abacus-2 datapath reuses its elements for multiple functions. The Manchester-carry path also serves as the shift-by-one and zero detect elements. The boolean unit sets up the carry path during add operations, functioning as the PG element.

Memory Organization The Abacus-2 memory will be organized as 32 rows by 16 columns. This organization requires only four column decoder wires, for a total of 20. Since the memory organization dictates PE organization, the PE aspect ratio will be much more square than the Abacus-1. Simulations show that the bitline discharge time is approximately 60% longer for the 16-cell bitline than for the 8-cell bitline, but the overall time is a small fraction of the entire read cycle.

1.2.3 Performance Estimation

Pin Budget. The chip package is an important aspect of overall design performance. To maintain low cost, the Abacus-2/8LC is housed in a 240-pin PQFP. The chip contains 64 8-bit PEs in an 8x8 arrangement. Each side has 16 pins for grid communication, sending data on both clock edges, or 32 bits per cycle.

Type	Quantity
Instruction	20
Mesh	76
External memory address	15
External memory data	32
External memory control	4
IO	8
TAP	5
Misc	5
Signal total	165
Power/ground	75

Table 1.1: Abacus 2 Pin Budget

Abacus-2 PE Area Estimation Sixty-four of these processors occupy approximately 30 mm^2, or about 75% of the area of the Abacus-1 array.

Cycle Time Estimation The limiting factor on the Abacus-2 ALU cycle is the propagation path of the manchester carry chain across 12 internal switches, 2 inter-slice switches,

Circuit	Area (uu^2)	Qty	Total (Kuu^2)
SRAM Cell	400	512	205
PG, sum, mux	6400	8	50
Pipeline	4000	8	32
Mesh Network	4300	8	35
Carry net,mux	3000	8	24
Aux Logic	10000	8	80
Total			425

Table 1.2: Abacus-2 PE Area

and 2 multiplexers (for a 16-bit add), followed by the sum calculation. This should take approximately 8 ns.

Dimensions	8 mm $\times$ 7 mm
PEs	64
Memory	32 Kb (512 bits/PE)
Pins	240
Technology	1 um CMOS
Voltage	2.5 V
Clock Rate	66 MHz
8-bit GOPS	4.2
16-bit GOPS	2.1

Table 1.3: Abacus-2 Chip Spec

1.3 Beyond SIMD: A Family of Abaci

Once a high performance SIMD core is developed, a number of interesting architectural ideas present themselves.

Abacus-M: Multi-SIMD Operation. One way of viewing the Abacus-2 chip is as the integration of a large number of traditional discrete bit-slices (such as the AMD2903) and a flexible interconnection network on a single chip. If bitslice family sequencer was also integrated, the chip could operate independently as a uni-processor, ignoring the broadcast SIMD instruction and treating the on-chip memory as registers and the off-chip memory as instruction and data store. This is essentially a superset of the recently common idea of integrating a RISC-style controller and with a SIMD array on chip.

The key difference is that several Abacus-M chips can work together as a larger SIMD system. This mode of operations entails dynamically selecting one of the chips as a master and then having all slaves listen on the instruction bus while the master drives the bus. The collection of Abacus-M chips could now operate in a mixed MIMD/SIMD mode, as required by the algorithm. This mode would not be as efficient as pure SIMD, but algorithms more suitable to MIMD operation could be implemented efficiently.

Abacus-S: Systolic Operation. A small augmentation of the Abacus-2 PE would allow operation in a systolic mode, where each PE executes a different instruction. The only required hardware is a locally controlled latch on every control line. Every PE could be now programmed to perform a different operation. For example, directing some PEs to multiply their network input by an internal constant and place the result on the network output, while others summed their network inputs, results in a pipelined FIR filter. A more powerful interconnection network is also necessary for effective operation in this mode.

Abacus-F: FPGA Emulation. The addition of local address decoding to an Abacus-S PE allows emulation of an SRAM-based FPGA cell. A richer interconnection network would again be required, but much of the network traffic could be time-multiplexed by a software-based routing scheme.

Abacus-U: Universal Computing Element. The combination of all three features described above leads to an almost universal computing element, able to operate as a SIMD node, as uniprocessor with a set of very wide vector registers, a collection of systolic computing elements, or as programmable logic. Furthermore, different parts of the chip could be operating in different modes. For instance, some PEs could be serving as logic gates configured as specialized find-first-one hardware co-processors for other SIMD-mode elements.