Architectural Trade-offs in a Latency Tolerant Gallium Arsenide Microprocessor

by

Michael D. Upton

A dissertation submitted in partial fulfillment

of the requirements for the degree of Doctor of Philosophy

(Electrical Engineering)

in The University of Michigan

1996

Doctoral Committee:

Associate Professor Richard B. Brown, CoChairperson

Professor Trevor N. Mudge, CoChairperson

Associate Professor Myron Campbell

Professor Edward S. Davidson

Professor Yale N. Patt

DEDICATION

To Kelly,

Without whose support this work may not have been started, would not have been enjoyed, and could not have been completed. Thank you for your continual support and encouragement.

ACKNOWLEDGEMENTS

Many people, both at Michigan and elsewhere, were instrumental in the completion of this work. I would like to thank my co-chairs, Richard Brown and Trevor Mudge, first for attracting me to Michigan, and then for allowing our group the freedom to explore many different ideas in architecture and circuit design. Their guidance and motivation combined to make this a truly memorable experience.

I am also grateful to each of my other dissertation committee members: Ed Davidson, Yale Patt, and Myron Campbell. The support and encouragement of the other faculty on the project, Karem Sakallah and Ron Lomax, is also gratefully acknowledged.

My friends and former colleagues Mark Rossman, Steve Sugiyama, Ray Farbarik, Tom Rossman and Kendall Russell were always willing to lend their assistance.

Richard Oettel continually reminded me of the valuable support of friends and family, and the importance of having fun in your work.

Our corporate sponsors: Cascade Design Automation, Chronologic, Cadence, and Metasoft, provided software and support that made this work possible. A design of this complexity would not have been possible using university-developed tools.

Significant development efforts were required in all areas of microprocessor design and computer architecture to bring this project to fruition. This work was not performed in isolation, and would not have been possible without the help of many others.

Aurora I: Dave Johnson wrote an initial Verilog to Cascade netlist translator. Rich Uhlig designed the Verilog RTL model. Ajay Chandna, Tom Huff, Tom Hoy and I designed the modules and layout floorplan.

Aurora II: Phil Barker and PJ Sherhart designed the cell layouts used for the Aurora II

chip. Taly Budescu assisted in all the horrible tasks no-one else wanted. Tim Stanley designed much of the bus interface and a behavioral MMU to allow the Aurora II model to run real code. Bob McVay modified GCC to produce code without byte operations.

Jim Dundas tested the scan chains of the final fabrication of the Aurora II chip, and Mark Roberts and Marie Powell tested the yield of the register files.

Aurora III: PJ Sherhart designed the bus interface unit of the Aurora III. David Kibler did the much improved cell layouts for the Aurora III chip. Dave Putti and Sara Domonkos designed an initial version of the Load-Store Unit. Bob McVay designed an initial version of the Instruction Fetch Unit. Tim Stanley reprised his role as behavioral MMU designer, providing crucial input into the design decision of the Aurora III bus interface unit.

A special thanks is reserved for Dave Putti, who single-handedly completed the final version of the Load Store unit for the Aurora III. Without his help this work would not have been completed.

TABLE OF CONTENTS

CHAPTER 7

CHAPTER 8

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
Introduction

A processor designed in a new technology must present significant performance and cost advantages over current design practices, or it will not succeed in the marketplace. New technologies are often advertised as having great advantages over those currently in production; however, the advantages often dissipate on further investigation. Much of the difficulty faced by a new technology is caused by rapid improvements in existing technologies.

A primary purpose of this work was to critically evaluate E/D MESFET Gallium Arsenide (GaAs) process technology for the design and implementation of high performance microprocessors. Technology has a profound impact on processor microarchitectures. The prospect of high transistor switching speed made GaAs Direct Coupled FET Logic (DCFL) an interesting technology high clock rate processor investigation. A second goal of this research was to investigate the effects ever-increasing clock rates would have on superscalar instruction issue capability.

A discussion of desirable process characteristics and technology trends will provide a context for the investigation of GaAs DCFL as a technology for microprocessor design.

1.1 Technology Requirements for High Performance Processors

There are two fundamental uses for transistors in a microprocessor: computation resources and state resources. To achieve good system-level performance, a technology must support both uses efficiently. The basic properties required for a high performance microprocessor include:

Logic Gate Switching Speed: The logical and arithmetic operations of a processor are implemented using logic gates to perform boolean operations on binary values. To achieve high performance the logic gates must be fast.

Circuit Density: The logic gates needed to implement a function should fit in a small area. Large area increases the interconnect capacitance, which slows the circuit and raises the power needed to switch these interconnect lines.

On-chip Memory: It must be possible to build memory structures on the processor chip. These memories are used both for register files and primary caches.

Interconnect Driving Ability: The technology must be able to rapidly drive signals that cross the chip. This means a high current buffer is needed.

I/O Bandwidth and Latency: Fast signal swing of external signals is important. Low voltage levels are needed to minimize noise generation.

Low Power Dissipation: Air cooled desktop processors require CPU power dissipations of 40 Watts or less.

Clock Distribution: Clock skew can consume a large percentage of the cycle time at high clock frequencies. The technology must support a low-skew clock distribution network. The on-chip frequency may be higher than the Input/Output (I/O) pin bandwidth, in which case a phase locked-loop clock multiplier is needed.

Based on these seven properties, Table 1.1 compares current technologies used in microprocessor design. Complementary Metal-Oxide-Semiconductor (CMOS) circuits dominate today's IC market. N-channel Metal-Oxide-Semiconductor (NMOS) technology was used extensively in the 1970's and early 1980's. Emitter Coupled Logic (ECL) has been in

Technology Attribute	CMOS	NMOS	ECL
Gate speed	Good	Fair	Excellent
Circuit density	Good	Excellent	Fair
On-chip memory	Good	Fair	Fair
Drive capability	Good	Good	Excellent
I/O bandwidth	Good	Fair	Excellent
Power dissipation	Good	Fair	Poor
Clocking	Good	Fair	Good

Table 1.1 Technology Support for Microprocessor Development

use since the 1960's and is still used in small, high frequency designs. CMOS technology has dominated recent designs, even though it excels at none of the required attributes. It provides a solid technology foundation, and unlike the other technologies has no fatal weaknesses.

By early 1990 Gallium Arsenide Direct Coupled FET Logic (GaAs DCFL) had reached a level of process maturity that would support chips of microprocessor complexity.

1.2 Microprocessor Density Increase Over Time

Since the introduction of the microprocessor by Intel in 1971, the continued advance of integrated circuit (IC) processing technology has allowed rapid and sustained growth in both the complexity and performance of single chip processors. Since the late 1960's, yearly lamentations of reaching fundamental limits have been raised.

Predictions for the continued improvement in technology have fallen between two extremes. In 1964 Thornton proposed the CDC 6600 system with parallel peripheral processors because it was felt that the technology of the time was reaching fundamental limits, and further increases in clock speed were not possible [Thornton70]. At the other extreme, In 1986, Sun Microsystems' Joy proposed the following estimate for processing power in Millions of Instructions Per Second(MIPS): $MIPS = 2^{year-1984}$. This equation predicts that processing power would double each year, beginning with 1 MIPS in 1984. The new Alpha 21164 processor from DEC does have a peak issue rate of 1.2 billion instructions per second. Although MIPS is a poor metric for comparing the performance of dissimilar processors, the Alpha could be seen as substantiating Joy's law.

In April 1965, Intel's Moore first observed that the complexity of integrated circuits was growing exponentially, doubling in transistor count every year [Moore79]. Many have since predicted this growth rate is destined to slow soon, only to be proved incorrect. It is now predicted that the final limitation on the achievable density will be economic rather than technological; fabrication facilities needed to produce ICs as dense as possible will

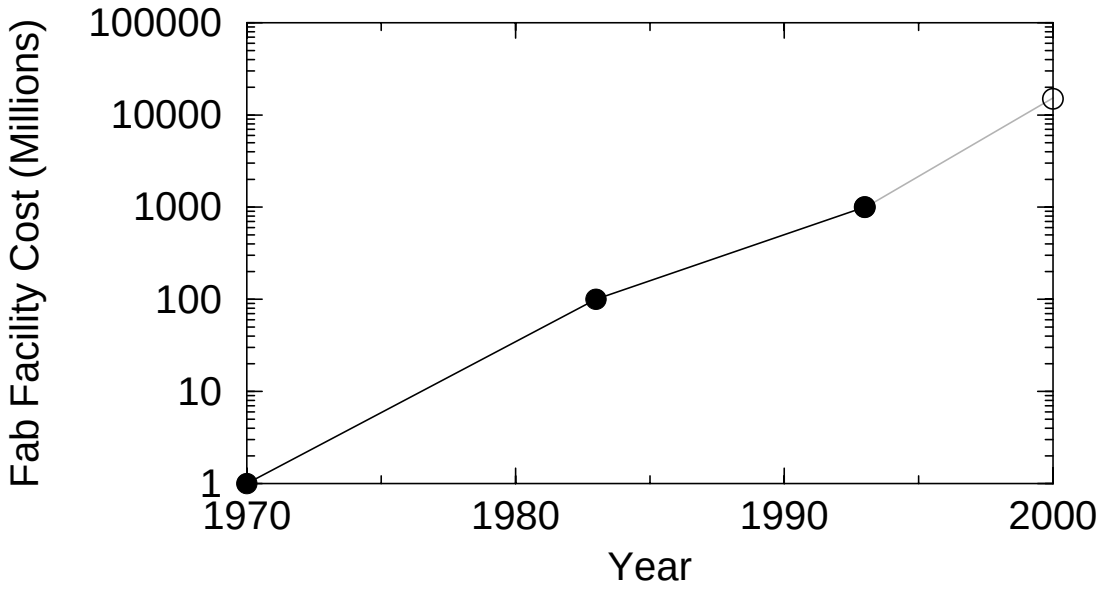

Figure 1.1 Fabrication Facility Cost

simply not be affordable [Koyima93]. Figure 1.1 plots the estimated cost of a new commodity IC fabrication line over a 30 year time span. In 1970, a typical fabrication plant cost around 1 million dollars, and each piece of fabrication equipment cost around $50,000 [Intel93]. Today, several companies are each spending over 1 billion dollars on new plants. The cost of fabrication equipment has risen to 3 to 4 million dollars for each machine, more than the cost of an entire fab in 1970! It is projected that a new fabrication plant will cost over 10 billion dollars by the turn of the century. Such high costs limit the number of companies capable of developing the next generation of fabrication lines, because the multi-billion-dollar investments required exceed the expected return on the investment for many products. It may be that only high-margin components, such as microprocessors, will be built on the most modern lines. The large increase in cost over the past two decades has serious implications regarding the ability of a new technology to succeed in the marketplace. However, the increase in plant cost has been offset by an even larger increase in plant efficiency. The cost per area of silicon has dropped continuously, due primarily to increases in throughput allowed by larger silicon wafers.

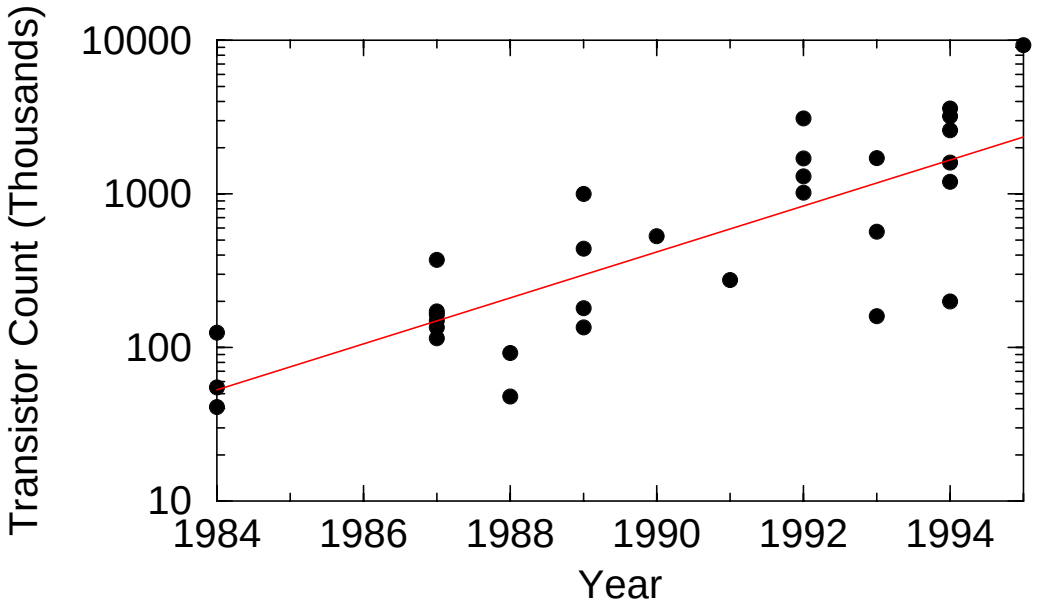

Figure 1.2 ISSCC Microprocessor Transistor Counts

The increase in fabrication plant efficiency has led to dramatic growth in the size of microprocessor chips. Figure 1.2 shows the total transistor count for processors introduced at the International Solid State Circuits Conference (ISSCC). There are three major components that have led to an increase in transistor count. First, the die size for microprocessors has grown rapidly, from 9 mm^2 for the 4004 in 1971 to 308 mm^2 for the Intel Pentium Pro processor in 1995 [Intel95]. Second, the smallest pattern capable of being reproduced on the integrated circuit, known as the feature size, has been dramatically reduced, allowing many more components to fit in the same area. The minimum feature size for the 4004 was 12 microns, and for the Pentium Pro it is 0.6 microns. Finally, newer fabrication processes provide additional metal routing layers, allowing much of the routing to occur over the top of the transistors, further reducing the area required to implement a given circuit. Together, these factors have caused a dramatic increase in the resources available for constructing processor chips.

The growth in fabrication capabilities has had a profound impact on microprocessor architecture. The continued increase in available resources has allowed many features previously seen only in large-scale computers to migrate to the desktop. Some of these features

include virtual memory, execution pipelines, caches, vector processing and support for fault tolerance.

As transistor counts have increased, computer designers have sought the most effective use of these additional resources. Microprocessors have devoted these extra transistors to two primary areas: additional on-chip cache memory, and parallel functional units for added computation capability. Increasing the size of the cache can absorb any amount of extra transistors, but Olukotun has shown that the optimal cache size may not be the largest possible [Olukotun92]. As the cache gets larger, the time required for decoding the address and sensing in the RAM array increases. If a single cycle is allowed for cache access, large caches can limit the maximum operating frequency of the chip, though the miss rate is reduced. Recent research has shown that a single cycle primary cache backed up by a larger multi-cycle secondary cache gives the highest system-level performance [Farrens94, Jouppi94].

The addition of on-chip floating point units is an example of using increased resources to provide additional computational power. Additional functional units perform computation in parallel with the work of the normal execution pipeline. In the extreme, the additional functional units may replicate much or all the execution pipeline, allowing the completion of multiple instructions each clock cycle.

1.3 Clock Speed Increase Over Time

As the number of transistors in microprocessors has grown over time, the clock frequency of the microprocessors has increased significantly as well. As the minimum feature size of IC processes has shrunk, the parasitic signal loading has reduced dramatically. The reduction in area has affected performance in two ways. First, smaller transistors switch faster and have lower gate capacitance, a major contributing factor to the switching speed. Second, the size of the major components in the microprocessors has been significantly reduced, greatly reducing the parasitic loading on most signal wires. The result has been a large increase in the operating frequency of microprocessor chips. Figure 1.3 shows the clock frequency of microprocessor chips presented at the ISSCC conference over the past

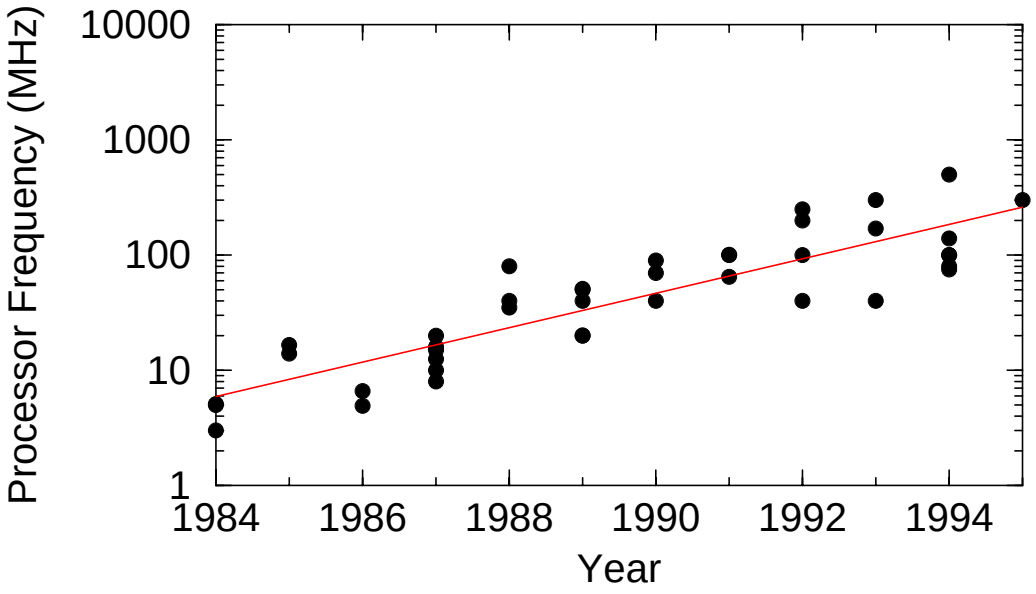

Figure 1.3 ISSCC Microprocessor Clock Frequencies

ten years. Individual processors are represented by points on the graph, and the solid line gives the best exponential fit to the data. Clock frequencies have been growing at a rate of 40% per year.

Although the raw computation frequency has risen dramatically, the system level performance has grown at an even faster rate. The most common measure of system level performance is the SPECmark, a measure of relative performance with a VAX 11/780 as the reference. Figure 1.4 presents integer SPECmark numbers for systems built from some of the chips described at ISSCC. The SPECmark numbers have been growing at a 59% yearly rate. In the past decade systems have improved in performance 100 fold.

1.4 Research Project Goals

This background of continued performance improvement sets the stage for the evaluation of GaAs processor design. When the GaAs processor project began in 1990, the fastest commercial systems were based on the 40MHz R3000 processor. At this time the goal of demonstrating a 250 MHz GaAs prototype appeared to be a significant advancement in the state of the art in processor design. Though the trends were obvious from Figure 1.2 and Figure 1.3, few people predicted the rapid increase in CPU operating frequency that has oc-

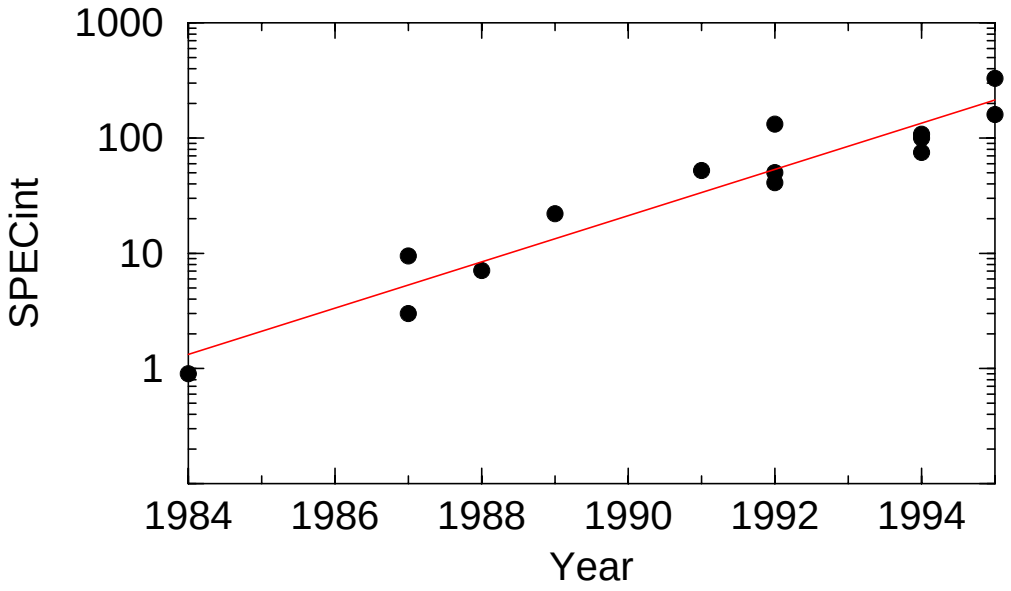

Figure 1.4 Integer SPECmark Performance

curred in CMOS processors. Current CMOS processors have far outstripped the initial goals of the GaAs processor project.

Processor performance growth has come from a combination of increased clock frequency and increased resources. Figure 1.5 shows how each component has contributed to SPECint performance. Clock frequencies have grown at 40% per year while the growth in SPECint has been 59% per year. As more resources have become available, the amount of

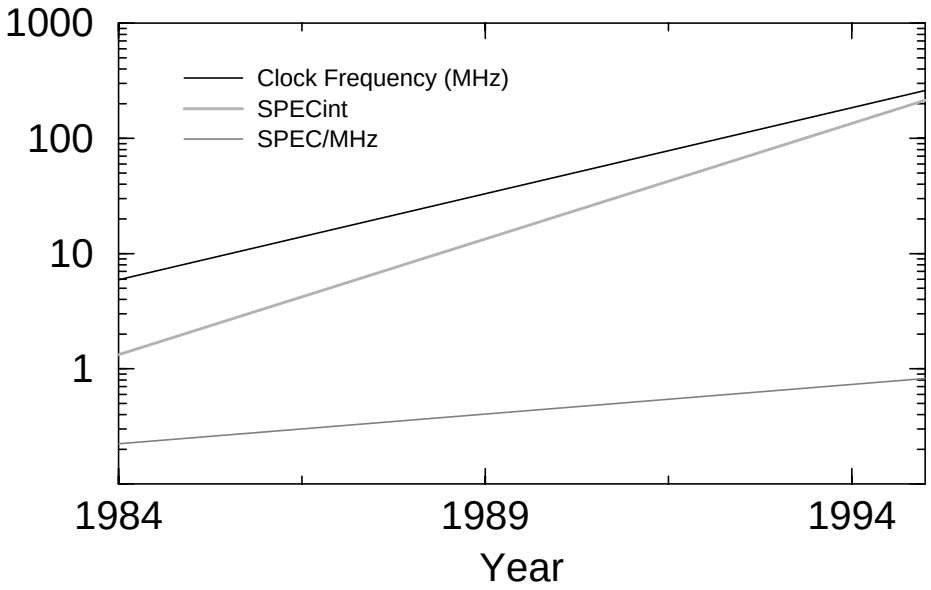

Figure 1.5 SPECint Performance vs. Clock Frequency Growth

work completed per clock has increased. This increase in efficiency is measured in SPEC/MHz, and has grown at a 13% annual growth rate from about 0.2 SPEC/Mhz in 1984 to an average of 1 SPEC/MHz in 1995. The increase in SPEC/MHz has primarily resulted from the addition of superscalar execution units [Johnson90].

One premise of this research is that it will become more difficult to exploit instruction-level parallelism as clock rates reach ever-higher levels. High Instruction Per Clock (IPC) levels are facilitated by low latency memory systems; building memory systems that provide low access latency, as measured by clock cycles, becomes difficult as CPU clock frequency is increased. This research focuses on developing a microarchitecture that can take advantage of moderate levels of instruction level parallelism without adversely affecting clock cycle time. Enhancements to the architecture to improve performance in the presence of long memory latencies are evaluated.

CHAPTER 2

The Importance of Latency Tolerance for High Clock-Rate Processors

2.1 CPU-Memory Performance Discrepancy

This chapter develops an analytical model for cache behavior, and uses it to characterize two different workloads. It is shown that memory system effects will dominate processor performance if memory architectures do not change. A set of modifications to the memory system are analyzed, and their use in several current microprocessors are highlighted.

In Chapter 1, Figure 1.4, it was shown that CPU performance has increased by a factor of 100 in the past decade. In contrast, main memory speeds have improved by less than a factor of ten in the same period, making it increasingly difficult to improve system performance. As fabrication process technologies improve, CPU cycle times are decreasing rapidly. The percentage of time the CPU sits idle waiting for memory is constantly growing. To compensate for the discrepancy in operating speeds, modern systems include cache memories near the processor that operate at processor speeds [Smith82]. Not all accesses can be serviced from the cache, so some fraction of time is still spent waiting for main memory. The CDC 6600 supercomputer, introduced in 1964, had a main memory cycle time of ten cycles of 100 ns for a total of 1000 ns. Main memory accesses on the 1978 vintage minicomputer DEC VAX 11-780 required six cycles of 200 ns for a total of 1200 ns [Jouppi90]. Current RISC processors have faster memory systems; the DECstation 3100 memory accesses complete in five cycles of 62.5 ns or 312 ns. The speed discrepancy is now beginning to widen severely with the advent of 100+ MHz machines. The HP 9000/735 requires 15 cycles of 10.1 ns or 155 ns for main memory access [Johnson91].

The VAX 11/780 would run at 60% of full speed even with the cache turned off [Jouppi90]. As processor speeds continue to increase, current-generation processors run at only a small fraction of their full speeds without their caches. With caches disabled, the HP

9000/735 slows down by a factor of more than 15. The growing discrepancy between CPU and memory speeds will have a profound impact on the future of computer architecture.

The continued increase in cache miss penalty causes significant performance degradation. Faster processors, with either higher issue rate or faster clock cycle, suffer a greater performance loss than slower processors. The increase in miss penalty reduces the amount of instruction level parallelism that can be extracted from programs. Cache miss behavior differs widely by workload, with database and operating system workloads exhibiting significantly poorer performance than other typical workstation applications [Cvetanovic94, Maynard94, Nagel94]. To maintain high performance with these workloads, processor designers must change the memory system.

Amdahl's Law

Amdahl developed a relation explaining the performance improvement in a system when the performance of only some of the components in the system are increased [Amdahl67]. Amdahl noted that the performance of the enhanced system is dependent on both the performance boost of the enhancement and the amount of time the enhancement can be employed: an enhanced system with little performance boost gives little benefit, and a large boost must be employed a majority of the time to give a substantial improvement. Amdahl's Law expresses this in equation form as derived by Hennessy and Patterson [Hennessy90]:

$$Speedup = \frac{Time_{original}}{Time_{enhanced}} \tag{1}$$

$$Time_{enhanced} = Time_{original} \times \left((1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}} \right) \tag{2}$$

$$Speedup = \frac{Time_{original}}{Time_{original} \times \left((1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}} \right)} \tag{3}$$

$$Speedup = \frac{1}{(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}} \quad (4)$$

To achieve the largest speedups the improvement must provide a large performance boost, and the improvement must be used a large percentage of the time. For example, if the improvement can only be applied one half the time, then Amdahl's law says the maximum speedup for the system with the improvement is two, even if the speedup of the enhanced system is infinite. Amdahl's law shows the performance of a system is often limited by its slowest component. As microprocessors become faster, the performance becomes limited by the time needed to process cache misses.

Current microprocessors typically have between 8K and 32K bytes of on-chip cache memory. Gee simulated the SPEC benchmark suite over a wide range of possible cache configurations using MIPS R3000 traces [Gee93]. Table 2.1 shows expected cache hit rates for direct mapped caches of various sizes with 32-byte lines for this technical workload. The table shows that for larger cache sizes very low miss rates can be attained. Similar results are obtained for other cache line sizes.

Different workloads can have significantly different behaviors. Maynard reports numbers similar to Gee for the integer SPEC benchmarks running on an RS6000 simulation environment [Maynard94]. She also gives numbers for large commercial workloads that have

Cache Size	I-Cache Hit Rate	D-Cache Hit Rate
1K byte	94.2	84.5
4K byte	97.1	92.3
16K byte	99.2	95.7
64K byte	99.8	98.3
256K byte	99.9	99.3
1M byte	100	99.7

Table 2.1 Integer SPEC Cache Hit Rates

much worse cache performance. Table 2.2 shows her numbers for the TPC A transaction processing workload for a direct mapped cache with 16-byte lines.

Commercial workloads have many processes and larger executables, resulting in low hit rates for the instruction side. The tendency in these workloads is to touch many data few times, leading to much poorer data cache behavior than for the technical workloads. The minimum D-cache miss rate plateaus at a few percent, and never reaches the 99+% range of the technical workloads.

Performance is a function not only of the miss rate, but also of the miss penalty. As processors become faster, primary cache miss penalties can grow from 10 to over 100 cycles. Furthermore, superscalar machines multiply the miss penalty by the issue rate, increasing the number of instruction issue slots that have been lost.

System Limited by Slowest Component

The speedup equation can be used to evaluate the effects of increased clock frequency on system efficiency. Performance is determined using the concept of an average instruction, as proposed by Emer and Clark [Emer84]. The model assumes single cycle execution of instructions, with 30 percent of instructions referencing memory. This gives an average instruction that has one cycle for execution, some number of cycles for processing instruction cache misses and additional cycles for processing data cache misses. The total cache miss rate for the average instruction is the I-cache miss rate plus 0.3 times the D-cache miss rate (because the average instruction has 0.3 memory operations). A system with 95% total cache hit rate, 20 ns cycle time and 200 ns main memory would have miss penalties of 10

Cache Size	I-Cache Hit Rate	D-Cache Hit Rate
8K byte	78.0	89.5
16K byte	82.0	92.0
64K byte	88.0	94.0
256K byte	96.0	95.5
1M byte	99.2	96.0

Table 2.2 TPC A Cache Hit Rates

cycles. If a scalar machine capable of completing one instruction each cycle is assumed, cache hits add no penalty, so 95% of instructions execute in a single cycle.

The 5% of accesses that miss in the cache require 10 cycles to access main memory. So the effective cycles per instruction (CPI) is:

$$CPI = Cycles_{execution} + Cycles_{Imiss} + Cycles_{Dmiss} \tag{5}$$

$$Cycles_{Imiss} + Cycles_{Dmiss} = (missrate_I + 0.3 missrate_D) \times misspenalty \tag{6}$$

$$missrate_{total} = (missrate_I + 0.3 missrate_D) \tag{7}$$

$$CPI = 1.0 + missrate_{total} \times misspenalty \tag{8}$$

For the configuration given above, this yields:

$$CPI = 1.0 + 0.05 \times 10 = 1.5 \tag{9}$$

Combined with the 20 ns cycle time, the 1.5 CPI gives a value of 33.3 MIPS. As described in more detail below, if the technology for the processor improved, reducing the cycle time to 10 ns, but the main memory access time remained the same, the miss penalty would double in processor cycles. With a 20 cycle miss penalty, the CPI would increase to 2.0, resulting in a processor performance of 50 MIPS. The 100% increase in CPU clock frequency would result in a system performance improvement of only 50%. According to Amdahl's Law, the maximum performance of this system could never exceed 100 MIPS, even if the processor were infinitely fast.

The equation for MIPS can be derived from Equation (8).

$$MIPS = \frac{1}{CPI \times t_c} = \frac{1}{(1.0 + missrate_{total} \times misspenaly) \times t_c} \tag{10}$$

Where t_c is the processor cycle time. Alternately,

$$MIPS = \frac{1}{(t_c + missrate_{total} \times t_{mem})} \tag{11}$$

Where t_{mem} is the main memory access time.

As cycle time reduces toward zero, the performance becomes dominated by the cache miss behavior of the system. The performance of the machine asymptotically approaches a peak MIPS set by the main memory cycle time and the cache miss rate. Thus, with an infinitely fast machine the peak MIPS is given by Equation (12).

$$MIPS_{peak} \approx \frac{1}{missrate_{total} \times t_{mem}} \tag{12}$$

Even with an infinitely fast processor, the 5% of accesses that miss still take 200 ns each, giving an average miss penalty of 10 ns. This results in the maximum MIPS rate of 100MIPS. At high processor clock frequencies very low miss rates are needed to ensure hardware resources are efficiently utilized.

Figure 2.1 shows the estimated SPECint performance for different cache sizes and cycle times. The miss rates are estimated from published results [Gee93], and assume a 32-byte line, direct mapped cache, 200 ns main memory access time, and an instruction mix that is 30% memory instructions. The cache model assumes near single cycle access times, either through a primary cache of the indicated size or a primary plus a low latency secondary cache. Advanced memory techniques, such as prefetching and nonblocking loads, are not accounted for in this simple analytical model. As will be shown Chapter 6, such techniques are needed to further improve system performance.

Several important relations are evident in Figure 2.1. First, the performance graph of slow processors is flat with increasing cache size, showing negligible performance improvement. As cycle times increase, the cache becomes a more important factor in performance. The performance plateaus at successively higher levels with larger cache sizes. At higher clock rates, more performance is gained by increasing cache size.

The estimated SPECint performance numbers indicate these benchmarks can reach high performance levels with high frequency processors and large caches. The model predicts that a 100 MHz machine could deliver 95 MIPS with a 128 K-byte cache, and a 1 GHz

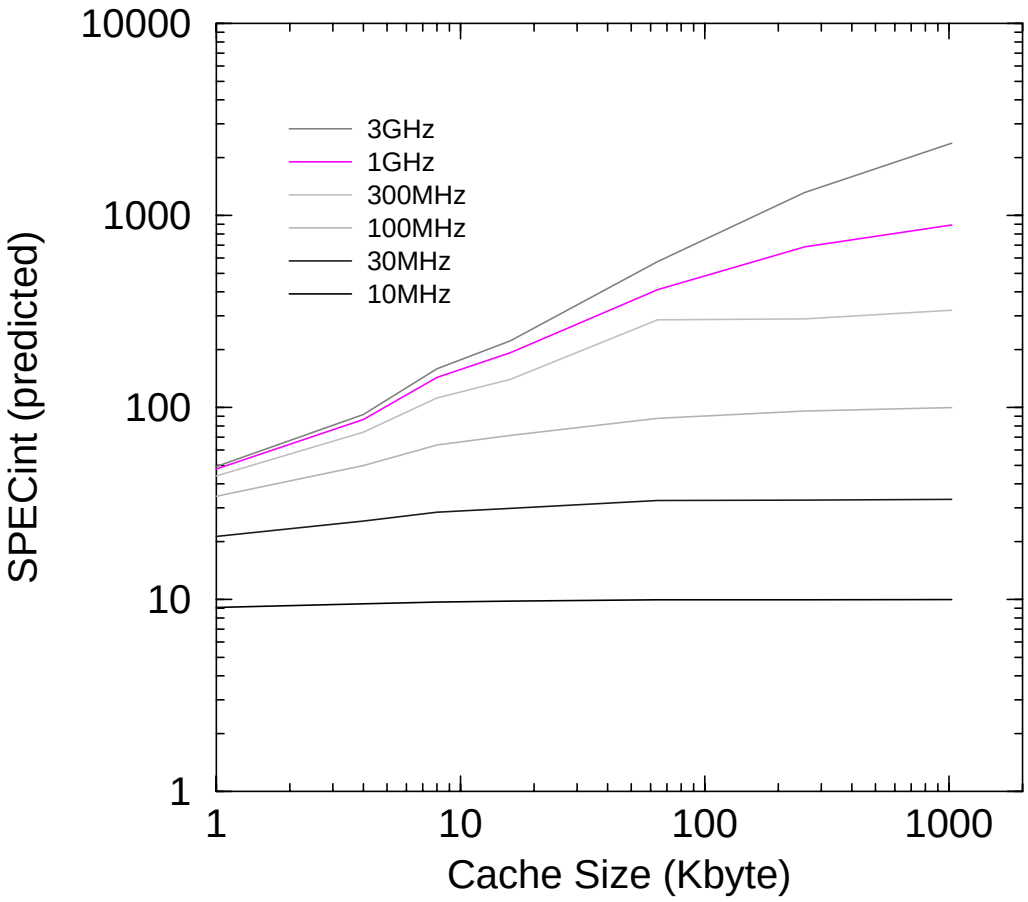

Figure 2.1 Estimated SPECint Performance

machine could reach 500 MIPS with the same size cache. However, the 1GHz machine would have lost one half of its potential performance due to cache misses.

Table 2.3 compares SPECint performance predicted by Amdahl's Law with actual SPECint performance numbers [SPEC93, Rubinfeld94] for several machine organizations. Considering the crudeness of the model (its inputs are cache miss rates, processor cycle time and main memory access time), the model predicts machine performance surprisingly well. The predicted numbers are quite close, except the R3000. The R3000 estimate is less accurate because, unlike other machines listed which achieve close to 1.0 SPECint/MHz average performance, older, single issue machines like the R3000 achieve only about 0.75 SPECint/MHz, so the model overestimates their performance. Much of the lost performance in such processors is in NOP instructions used to pad out load-use delays and branch slots. On average, about 20% of the instructions executed are NOPs; this would account for

Processor	Clock (ns)	Cache (int,ext)	SPECint Prediction	SPECint Actual
R3000	30	0,64K	32	20.9
Pentium	10	8K,256K	95.5	100
Alpha 21064	5	8K,1M	195	170
Alpha 21164	3	96K,4M	320	330

Table 2.3 Predicted SPECint Performance

about half of the discrepancy. Wide issue, superscalar machines like the SuperSPARC would have their performance underestimated by the model, since they are able to complete more than one instruction per cycle. Nevertheless, the model is useful for illustrating the relationship between clock speed, memory characteristics and system performance.

Using the analytical cache model, the amount of cache necessary at each clock frequency to achieve 50% of maximum performance can be calculated for the SPEC benchmarks. This value, called the $C_{1/2}$ value and is analogous to the $N_{1/2}$ value that measures the vector length needed to achieve 1/2 the peak performance in a vector machine [Hennessey90].

Figure 2.2 shows that the growth in the $C_{1/2}$ value with increasing CPU clock frequency is geometric. A machine with a 100MHz clock needs only 4K of cache to achieve 50% of maximum performance. As clock frequencies near 1 GHz, 128K-byte of low latency cache is needed to maintain performance at 50% of maximum levels. A 1GHz machine like this would deliver 700SPECint.

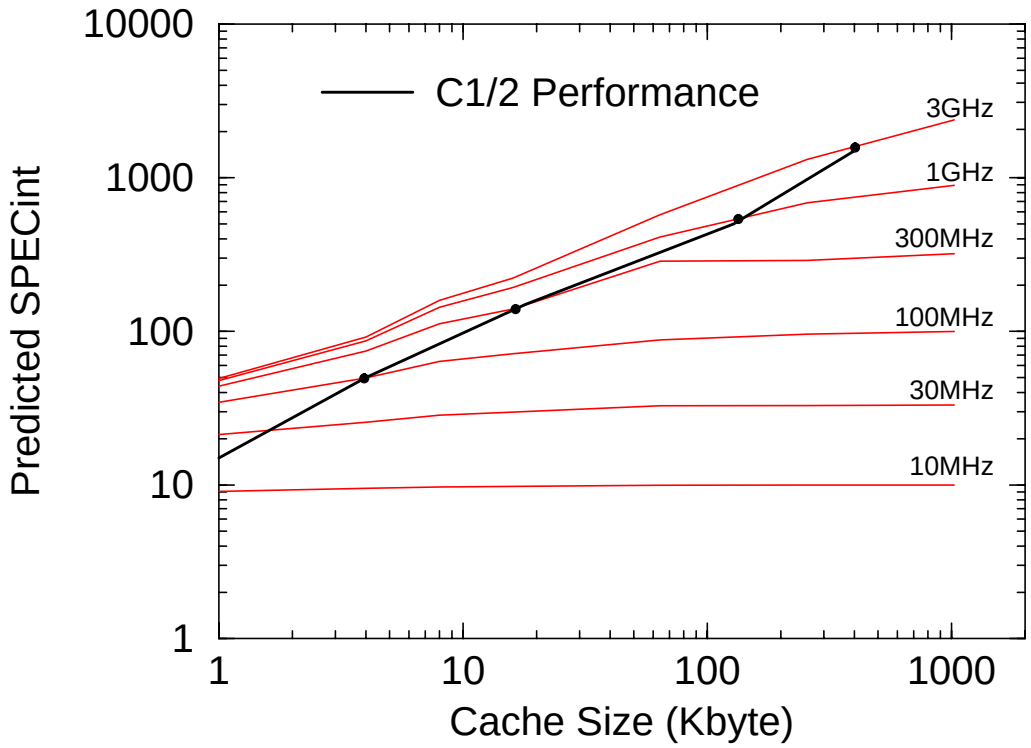

Figure 2.2 Cache Size Resulting in 50% of Peak Performance

Similar predicted performance graphs can be generated for other workloads. Figure 2.3 shows relative performance improvements that can be obtained with various cache sizes and clock frequencies on the TPCA benchmark. The estimated performance for this workload is much more modest than for the SPECint workload.In fact, with less than 128K-bytes of cache, the performance improves little for clock frequencies above 100MHz.

2.2 Cache Miss Characterization

Caches work by exploiting both spacial and temporal locality. Instruction and data accesses have different behaviors; tuning the parameters for the different caches can provide a substantial performance boost. In general, instruction stream accesses exhibit more spatial locality than data accesses. Cache misses can be grouped into three broad categories: Compulsory, Conflict and Capacity. Different design solutions are needed to take advantage of the different classes of cache miss.

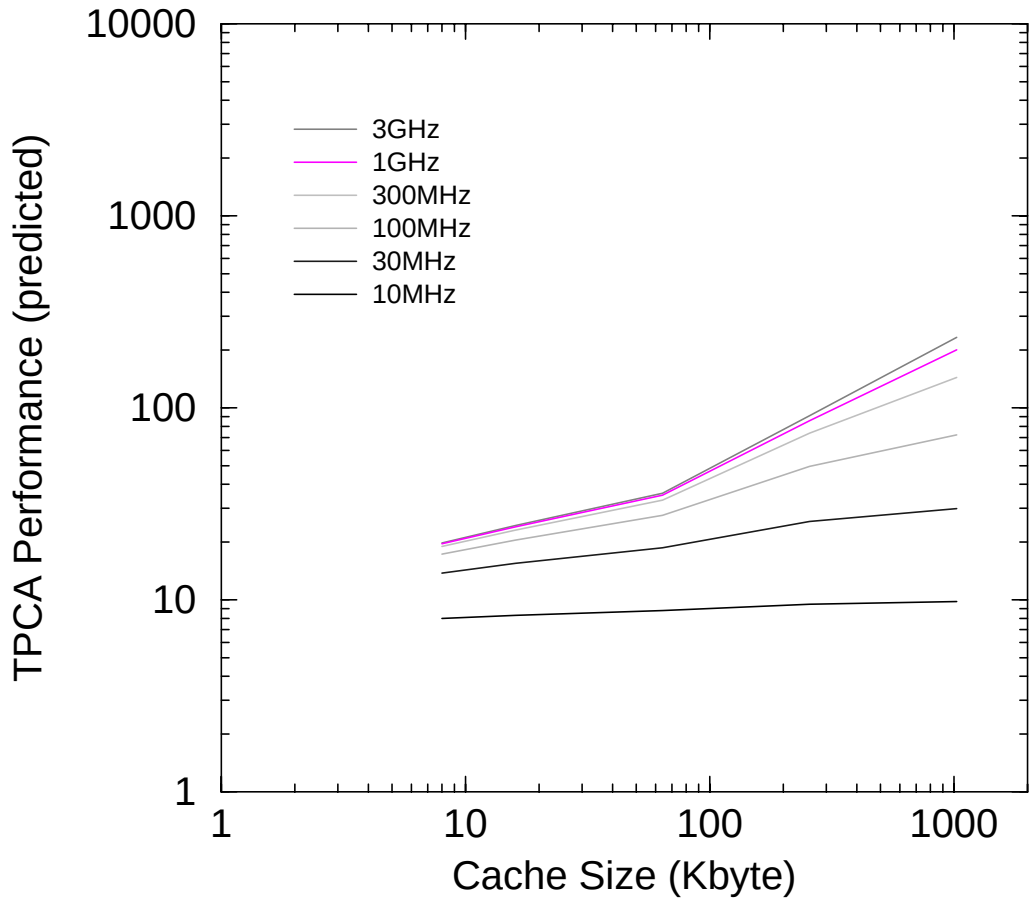

Figure 2.3 Relative TPCA Performance

Compulsory Misses

A compulsory miss occurs the first time a memory location is referenced. Because no previous reference has been made there is no way the cache could hold the data, and a miss results.

Memory accesses are often spatially clustered, especially the instruction accesses. If a word has been accessed, it is likely that a nearby word will be accessed. This property can be exploited by increasing the line size of the cache and fetching multiple words on each cache miss. In a cache having a longer line size, only the first access to the line causes a miss. This is especially useful for instructions, because most accesses are sequential. Except for control flow instructions, the next address is guaranteed to be the next sequential memory location. For small values of N, increasing the line size by a factor of N reduces the compulsory miss rate by nearly N [Smith82].

There is a limit to how much the line size should be increased. Increasing the line size reduces the number of lines in the cache, slightly increasing the miss rate. Not all words in a cache line will be accessed; if the line becomes too long a substantial fraction of the typical line will never be used. Fetching the unused portion only adds overhead to the time needed to access the required location. Lengthening line also increases the time needed to fetch and fill the line in the cache; and increases the bandwidth needed from the memory system.

Conflict Misses

Conflict misses occur when two or more memory locations map to the same entry in the cache. There are many possible ways to map data in a cache. The most flexible mapping is known as fully associative. In a fully-associative cache, each cache line is free to occupy any cache location. A content-addressable tag memory determines which cache entry holds a requested data item. At the other extreme of mapping freedom is a direct-mapped cache. In a direct-mapped cache, each item can map to only one possible cache location. Set-associative caches lie between these two extremes in mapping freedom.

Mapping conflicts cause increased misses when compared to fully-associative caches Conflict misses can be reduced by increasing the associativity of the cache. Direct-mapped caches suffer the most from conflict misses, and can even suffer pathological access patterns where the cache hit rate drops to zero.

Cache hit rate will increase with increases in associativity, due to the reduction in conflict misses. However, most of the performance benefit of increased associativity can be achieved with low degrees of set-associativity, such as 2 or 4 way set associativity. The added time needed to select the proper set may extend the critical path, removing any performance benefit gained by higher hit rates [Hill88].

Capacity Misses

Capacity misses occur when the working set size of the data exceeds the size of the cache. If the working set is too large, data must be evicted from the cache to make room for the newly requested items. This is required regardless of the associativity of the cache.

When this data is accessed again, it will be brought back into the cache, displacing some other item. If the working set is too large to fit in the cache, the only solution is to increase the cache size.

2.3 Memory system enhancements to maintain performance

The memory model used in the performance estimation in Section 2.1 assumed two restrictions, the removal of which can significantly improve processor performance. The first restriction is that memory cannot be accessed before it is requested by the processor. The second is that all cache accesses are processed serially. Prefetch hardware removes the first restriction and nonblocking loads remove the second restriction.

2.3.1 Prefetching

One way to maintain high performance in a deeply-pipelined machine with a high clock rate is to eliminate cache misses. Prefetching brings data on chip before it is referenced by the processor. When prefetched data is referenced, the data can be quickly supplied, eliminating the long latency associated with an off-chip access.

Software Prefetch Methods

Mowry investigated how prefetch instructions can be inserted into scientific code by the compiler [Mowry92]. This method works well for loop-based programs with regular and predictable access patterns, such as data accesses in scientific programs.

There are problems with this method when the end of the loop is reached. The inserted prefetch instructions must not access data outside the allowed array range. To do so could cause memory access exceptions and poor performance. The main drawback to software prefetch methods is that they cover only a subset of possible applications, and it is difficult to predict the access patterns of integer code [Klaiber91, Chen94].

While high performance can only be achieved through a combination of hardware and software methods, this thesis is primarily concerned with hardware solutions for improving performance.

Prefetch into Cache

Prefetching can be built entirely in hardware, without software support [Fu92]. The prefetched instruction or data can be written into either the primary cache or to an auxiliary memory buffer. However, there are drawbacks to both approaches. Prefetched items will not always be used; unused items increase bus traffic without improving performance. Prefetch items written into the cache may displace real instructions or data that will be needed by the application, thus causing additional cache misses. Pierce has shown that this is usually not a problem for the instruction stream [Pierce94]. In the other case, when prefetched items are written into an auxiliary memory rather than into the cache, extra cycles may be needed to route the data or instructions from the auxiliary memory to the execution pipeline.

Stream Buffers

Jouppi has proposed a hardware-based prefetching scheme called the stream buffer [Jouppi90]. A stream buffer contains an address tag, a tag comparator, an incrementor and a few cache lines of data. Figure 2.4 is a diagram of a typical stream buffer organization.

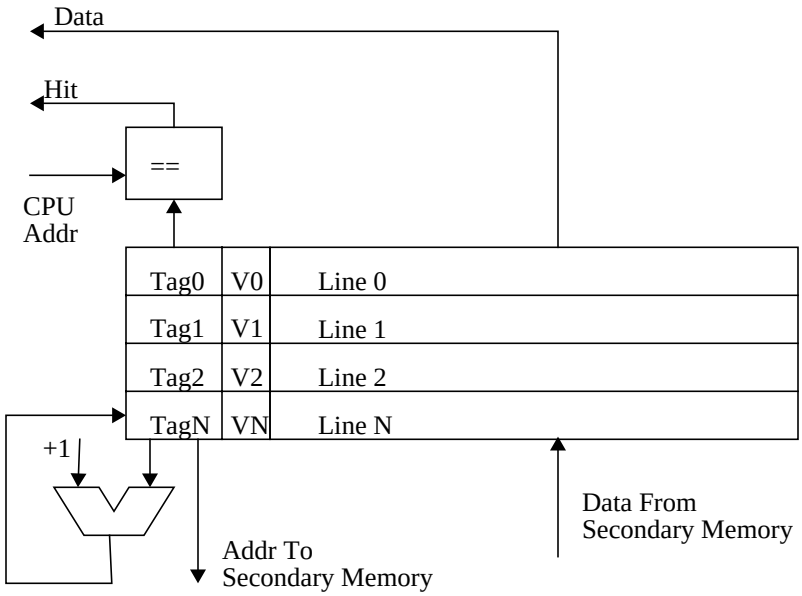

Figure 2.4 Stream Buffer Organization

The hardware resources needed are modest, the primary cost being the memory needed to buffer the cache lines.

After a cache miss, the address of the next cache line is loaded into the stream buffer line 0 entry. That line is requested from the secondary memory system, and when the data returns, the line is marked valid. Subsequent cache misses compare the requested address with the tag stored in line 0. If the request matches the address tag, the data has already been requested, and can be sent to the requesting unit when it arrives from secondary memory. After supplying the data to the requesting unit the stream buffer pops the top entry from line 0, moving the successive entries up one position.

With long-latency memory systems, it may not be possible to eliminate the fetch latency prefetching single cache lines. Additional cache lines can be fetched during times when the memory bus would otherwise be idle, allowing the fetch address to get many lines ahead of the currently requested address. To minimize cost, only one comparator is used; addresses must advance to line 0 before they can be matched.

Prefetching allows smaller I-cache

I-cache miss rates can be substantially reduced with the addition of even a small prefetch buffer. Jouppi's results show that a stream buffer having only a few entries can often provide the same performance as doubling of the instruction cache size [Jouppi90]. Because of the sequential nature of instruction execution, about 70% of accesses to new cache lines reference the next sequential cache line.

The primary drawbacks to prefetching are the added complexity and memory bandwidth needed. Not all the memory requested by hardware prefetching will be used. If too much additional prefetching traffic is generated, the prefetch instructions and data will interfere with the normal cache miss memory traffic, possibly reducing performance.

2.3.2 Nonblocking memory operations

A primary assumption in the performance estimates of Table 2.3 is that cache misses are processed serially. This leads to a large amount of dead time where instructions are not

issued. In reality, cache misses often appear in closely associated groups, often to the same cache line. Kroft proposed a method for allowing multiple cache misses to be processed in parallel [Kroft81]. He noted that if processing could continue after a cache miss, it would be possible to detect additional misses and begin their servicing in parallel with the misses already in progress. This method is known as nonblocking-load or lockup-free caching.

This technique typically uses a small number (usually between two and eight) of nonblocking load registers. Kroft's results showed that his scheme rarely needed more than three nonblocking load registers. This is, of course, dependent on the processor issue model.

In an in-order issue machine, processing continues after the first cache miss until an operation that uses the result of the load is detected. With more sophisticated out-of-order-issue machines, processing can continue even past these dependent instructions until all available resources have been consumed. An out-of-order issue machine will stall when there are no more unexecuted instructions in the instruction queue, or when there are no more nonblocking load registers available to queue a nonblocking load request.

2.4 Elements of Current Microprocessors

Improvements in microprocessor performance have had two primary sources, technological improvements and architectural improvements. In 1980, microprocessors were not pipelined, contained no on-chip cache and required multiple cycles to complete each instruction. The increased transistor resource budgets of recent years have allowed each of these performance limitations to be corrected.

This section reviews some recent microprocessors to see how they address issues in high speed design; the use of instruction level parallelism, in particular, is described. Methods for maintaining a high clock rate will also be discussed. Current microprocessors can be characterized by issue width, number of memory operations performed each cycle, issue model, completion model and degree of nonblocking memory operations supported.

2.4.1 R4400

The R4400 is the latest 64-bit R4000 family processor from the MIPS technology division of SGI. It is functionally the same as the R4000, but with larger on-chip caches, 16K each for instructions and data. The R4400 is a single issue, superpipelined design, with an eight-stage pipeline [Heinrich94]. The processor emphasizes clock rate over parallel issue, and operates at up to 200 MHz. Cache accesses take two internal CPU cycles. Because of the long cache access delay, a significant number of cycles are lost to branch delays and data dependencies. Hardware load interlocks are necessary to ensure correct operation because the load delay is longer than a single cycle. The chip issues and completes instructions in order, and has no branch prediction capability, but does add several instruction set extensions to minimize the impact of branch delay slot NOP instructions.

2.4.2 R8000

The R8000 TFP processor is a two-chip decoupled implementation of the R4000 architecture [Hsu94]. The processor runs at a relatively slow 75 MHz, but the integer and floating point units are capable of executing up to 4 instructions each cycle, for a maximum throughput of 300 MIPS or 300 MFLOPS. No more than four instructions can issue in any cycle. The basic R4000 instruction set was refined to add an IBM multiply-add style instruction that is useful in vector codes.

The processor features a unique second-level cache architecture where all floating-point data is cached only in a large, pipelined secondary cache. Integer data and all instructions reside in separate 16K-byte caches on the integer chip. The integer and floating point units communicate via a set of interface queues. To speed instruction fetch, branch prediction is integrated into the primary instruction cache hardware. Each cache line contains the predicted next cache index, which can be used immediately to fetch the next cache line; no branch offset calculation is required.

2.4.3 R10000

The R10000 (also known as the T5) is the next processor to be introduced by the MIPS

Technology division of SGI. This processor is a significant departure from the R4400, with much wider issue and out-of-order instruction issue and completion capability. The R10000 fetches four instructions per cycle and issues them to separate queues for integer, memory, and floating point operations. To make best use of available memory bandwidth, the processor supports up to 4 outstanding cache misses. Unlike many other processors, however, the R10000 does not stall when an instruction references the result of a pending load. Instead, it scans the queues for other memory instructions that are ready to issue, and issues these instructions out of program order while waiting for the requested results.

The processor uses register renaming with a pool of 64 physical registers to map the 32 logical registers. When an exception or branch miss occurs, the previous machine state can be quickly recovered by resetting the register mapping information to a previously saved value.

2.4.4 SuperSPARC

The SuperSPARC (also known as Viking) processor was the fastest 32-bit SPARC processor from Sun Microsystems in 1994. It emphasizes instruction level parallelism over clock rate, issuing up to three instructions simultaneously from a four-instruction window.

There are two primary manifestations of the emphasis on parallelism. First, the processor can perform two sequential ALU operations each clock cycle. The processor contains three ALUs, arranged in a tree.The first level of the tree has two ALUs in parallel, each of which can accept an ALU operation if the two operations are independent. If the second operation is dependent on the result of the first, the first result can be fed directly to a third ALU positioned after the first two. Second, there is no delay after a load instruction before the data can be used. This requires that a complete cache access must be completed within one clock cycle. The result of these decisions is that more parallelism is available, but at a price in clock frequency and performance.

The SuperSPARC processor achieves one of the highest SPECmark ratings for a given operating frequency. However, its features dramatically reduce the maximum operating frequency of the processor. The chip was originally designed to operate at 50 MHz, but vol-

ume shipments were originally only available at 36 MHz. This speed was less than half that of its contemporary processors.

2.4.5 UltraSPARC

UltraSPARC corrects most of the poor design decisions made in the SuperSPARC. The cascaded ALUs have been eliminated in this quad-issue chip, and the cache access no longer sets the processor operating frequency.

Like the R8000, the UltraSPARC caches the predicted next address field for each cache line, and tags each cache line with bits to indicate the branch type, allowing branches to be issued each cycle with no address calculation delay. A unique feature of the UltraSPARC is the instruction set support for image processing primitives. Like the Motorola 88110, UltraSPARC defines a set of graphics instructions and data types that can be processed in Single Instruction Multiple Data (SIMD) function units. Although the chip can issue up to four instructions in a cycle, the fourth instruction must be a branch or floating point instruction.

The processor supports nonblocking loads; up to nine loads and eight stores can be pending. Support for a large, pipelined secondary cache is contained on chip. Because of the large size of the chip, 3.8 million transistors, the process has been switched from the BiCMOS used to fabricate the SuperSPARC to a standard CMOS process.

2.4.6 Intel Pentium Processor

The Pentium Processor contains a pipeline constructed for the efficient execution of the X86 instruction set [Schutz94]. This processor is a good example of how to build a fast pipelined CISC processor. The Pentium processor executes in hardwired logic a subset of the X86 architecture, and implements the remainder in microcode. Most simple operations execute in a single clock cycle, and two instructions can be issued each clock if there are no instruction dependencies and the instructions are of the proper types to be paired.

To support two simultaneous data accesses in parallel, the data cache is divided into 8 banks. If two requests map to different banks, the accesses can proceed in parallel. Since bank conflicts are rare, this design provides nearly the performance of a dual-port cache at

much lower cost. The processor issues and completes instructions in-order.

2.4.7 AMD K5

The AMD K5 processor is a superscalar implementation of the X86 architecture [Halfhill94]. Unlike the Pentium processor that features an in-order dual-issue CISC pipeline, the K5 dynamically translates the X86 instruction stream into fixed-length RISC-like instructions, that are executed on an out-of-order quad-issue RISC pipeline.

This design process made extensive use of CAD tools. The datapath blocks were full custom layouts, but much of the control logic was generated by logic synthesis tools from a behavioral language description and placed and routed automatically.

Instructions are flagged in the I-cache to determine the boundaries of variable length instructions. Instructions are issued to a set of reservation stations at each function unit, where they may be executed out of order. Out-of-order completion with precise exception resolution is implemented using a 16-entry reorder buffer. Instructions complete out of order into the reorder buffer, and are then written from the reorder buffer into the register file in sequential order. Like the UltraSPARC and R8000, the K5 uses in-cache next address caching to predict branches. The K5 also uses a 1 bit dynamic branch prediction.

2.4.8 Motorola 88110

The 88110 is the last processor of the 88K family from Motorola [Diefendorff92]. It has a regular instruction set and dual-issue capability. The 88110 has ten independent functional units, including two integer ALUs and two graphics units. The graphics units perform SIMD operations on many pixels at once. The 88K architecture specifies a unified integer and floating point register set, which severely restricts the design options for a superscalar machine, and the 88110 designers expended significant effort working around this architectural bottleneck, adding a second complete register file for the floating point operands.

Most simple operations in the 88110 complete in three clock cycles using a 4-stage pipeline. Loads and floating point operations require additional cycles. A register file scoreboard [Thornton70] keeps track of when instruction operands are available, and de-

lays issue of an instruction until all of the input operands are ready.

A history buffer is used to record the old register contents so they can be restored after an exception. This method is viewed as inferior to other methods such as register renaming and reorder buffers used on more modern designs, primarily because of the time required to restore the machine after an exception or branch misspredict[Diefendorff92]. Exceptions cause little problem because they are rare, but restoring the state after branch mispredictions results in excessive overhead. Mispredicted branches are quite common, causing many cycles to be lost to state recovery using the history buffer.

A high-speed system bus supplies data for the 8K-byte on-chip instruction and data caches. The split transaction bus is 64 bits wide and supports bus snooping and critical-word-first requests. Although the 88110 is the last 88K processor, the system bus is being used on future Motorola PowerPC processors.

2.4.9 IBM/Motorola PowerPC 604

The PowerPC 604 is a high-end 32-bit PowerPC chip [Song94]. It makes extensive use of reservation stations, out-of-order issue, and out-of-order completion to achieve a high execution rate at a relatively low clock frequency.

The PowerPC architecture is an extension of the IBM RS6000 architecture, and features multiple register sets for integer, floating point and conditional operations. The separate register sets allow instruction fetch processing to be separated from the rest of the integer execution unit; all program counter processing is performed by a separate instruction fetch unit. Separating the processor into multiple independent units exposes more instruction level parallelism.

The PowerPC 604 supports out-of-order instruction issue, and uses a reorder buffer and rename registers to maintain precise exceptions. The memory system is nonblocking, supporting up to four concurrent cache misses. The system I/O uses a split transaction bus to overlap multiple bus transactions.

2.4.10 DEC Alpha 21064

The first DEC Alpha processor, the 21064, was announced at the ISSCC conference in 1992 [Dobberpuhl92]. With its 200 MHz clock speed and 64-bit architecture, this processor lead the competition in performance. The Alpha processor has several unusual design features, due primarily to the requirement that the design should be able to scale in performance over 25 years. The design is clean, with few first implementation artifacts. The machine is superscalar, but with restrictive issue rules. Two instructions may issue each cycle, but the instructions must be from different classes: integer operation, floating point operation, branch or memory.

Some of the unusual design attributes include the lack of byte load and store instructions and the maintenance of precise exceptions through software synchronization instructions. Another unusual feature of the 21064 is the clocking scheme. A true-single-phase clock scheme is used with a single global clock wire driven by a very large clock buffer. One phase uses latches that are transparent when the clock is high, and the opposite phase latches are transparent when the clock is low. This scheme is only possible in complementary technologies. The clock buffer alone dissipates 9 W!

2.4.11 DEC Alpha 21164

The current performance leader is the next DEC Alpha chip, the 21164 [Rubinfeld94]. This chip achieves its performance through a high clock rate, simple quad-issue model and large two-level on-chip cache. The two-level cache was adopted to reduce the latency to the smaller first-level instruction and data caches. The 8K-byte size of the primary caches allows them to be accessed in a single cycle, one cycle faster than in the 21064. The smaller cache latency greatly reduces stall cycles. To provide low miss rates, the fast primary caches are backed up by an on-chip 96K-byte three-way set-associative secondary cache. Another benefit of the two-level on-chip cache hierarchy is that the primary data cache is small enough to be fully dual ported, allowing simultaneous access from both integer pipelines. The chip includes hardware prefetch support for instructions; the processor can run at full speed from the secondary cache with the help of these buffers.

Processor	Date	Tran Count	Issue Width (I.M.F)	Issue Model	Completion Model	Non-blocking Load
R4400	1993	2.2M	1(1.1.1)	In-Order	In-Order	No
TFP	1994	2.6M	4(2.2.4)	In-Order	In-Order	No
T5	1995	6.8M	4(2,1,2)	Out-of-Order	Out-of-Order	Yes
SuperSPARC	1992	3.1M	3(2,1,1)	In-Order	In-Order	No
UltraSPARC	1995	3.8M	4(2,1,2)	In-Order	Out-of-Order	Yes
Pentium	1993	3.1M	2(2.2.1)	In-Order	In-Order	No
K5	1995	4.3M	4(2.2.2)	Out-of-Order	Out-of-Order	No
88110	1992	1.3M	4(2.1.2)	In-Order	Out-of-Order	No
604	1994	3.6M	4(3.1.1)	Out-of-Order	Out-of-Order	Yes
21064	1992	1.68M	2(1,1,1)	In-Order	In-Order	No
21164	1995	9.3M	4(2.2.2)	In-Order	In-Order	Yes

Table 2.4 Current Processor Characteristics

The 21164 is an in-order issue, in-order completion machine. To counteract the effects of growing memory latency, the processor supports nonblocking loads. Up to six primary data cache line fills can be pending, with as many as 21 load instructions waiting for the data. Instruction issue continues until an instruction references the result of an outstanding load miss.

Table 2.4 summarizes the features of the different machines described previously. The capabilities have progressed from scalar, in-order machines to superscalar machines supporting multiple memory references each cycle and nonblocking loads and out-of-order completion. While the latest machines are quad issue, only the PowerPC 604 supports more than two simultaneous integer operations. The other machines support issue classes, and require that the instruction stream be of the proper class mix to maintain maximum issue rate. Only two machines have added out-of-order instruction issue, but all recent machines have added support for nonblocking loads or out-of-order completion.

2.5 Computational Efficiency and Delivered Performance

Increasing processor clock rates without increasing memory speed proportionally re-

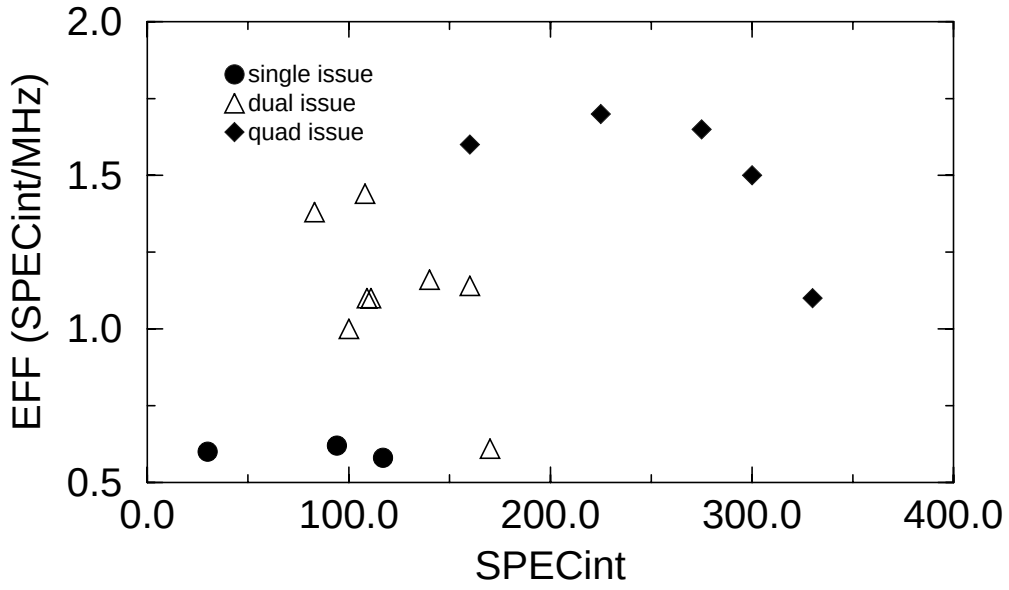

Figure 2.5 Correlation of Efficiency and SPECint Performance

duces the benefit of multiple instruction issue. Figure 1.5 showed the contribution of IPC and clock frequency to SPECint performance. Average instruction issue rate has been shown to track the SPECint performance of a processor divided by the clock frequency of the processor (SPECint/MHz). To date, clock frequency has been the major contributor to performance. Will this continue to be the case in the future?

Figure 2.5 shows the changes in performance and computational efficiency as measured by SPECint/MHz. Three machine types are shown: single issue, dual issue and quad issue. Three distinct regions of operation are seen in the figure. Single issue machines achieve nearly constant performance irrespective of architecture. These machines get about 0.6 SPECint for each MHz of clock frequency. The most effective way to speed up a single issue machine is to speed up its clock.

There is no clear optimal design for the dual issue machines. The performance of the dual issue machines is not correlated with the computational efficiency; based on the data points in Figure 2.5, both flexible instruction issue and higher clock rates are equally valid methods of improving performance of a dual-issue processor. Simpler machines can be brought to market faster than complicated machines.

With the recent announcement of the next generation processors from several vendors, there is now sufficient data to speculate on quad-issue machine design issues. The clock rates for these machines continues to climb. The increased speed mismatch between the processor and memory system has reduced the ability of these processors to take advantage of additional parallel execution units. The quad issue machines have twice as many execution unit as the dual issue machines, but achieve a SPECint/MHz only 30% better than the dual issue machines. To compensate for increasing memory system effects, all the quad issue machines have added nonblocking memory operations.

The performance data for these machines indicates that efficiency and performance are negatively correlated. The highest performance machine in each issue group is the one with the highest clock rate, and the lowest execution efficiency. Attempting to improve performance by increasing efficiency appears to slow the clock more than it improves the SPEC rating, resulting in lower performance for more complicated machines. The highest performance machine is the Alpha 21164, with a 300 MHz clock and an efficiency of 1.1 SPECint/MHz.

2.6 Summary

Current microprocessors look remarkably similar to each other. There is a major emphasis is on efficiently exploiting the instruction-level parallelism that exists in scalar code, and maintaining a high throughput rate in the memory system. Today, a primary goal of microprocessor systems is to keep the memory system constantly busy, and to overlap cache miss processing using nonblocking memory operations.

Microprocessor Report has grouped the architectures into two differing camps: speed demons and brainiacs, distinguished by how they attempt to exploit existing parallelism [Gwennap93]. Gwennap described speed demons as machines that focus on more streamlined architectures offering less flexibility but higher clock rates. The brainiacs run at a slower clock rate but are capable of more computation each cycle. A still open question is which method is superior, but according to the data in Figure 2.5, faster clock rates and simpler issue models result in faster machines.

Current microprocessors are adding increasing complexity to the memory system to minimize the increasing performance degradation caused by cache misses. Caches are getting much larger, both as large single-level primary caches and in multilevel on-chip cache hierarchies. Hardware prefetching is added to the instruction cache to fetch instructions before they are requested by the processor, minimizing the stall time. Non-blocking memory accesses allow multiple cache misses to be processed in parallel, reducing the average overhead. The machines are making increased use of queues to decouple multiple independent function units. Using these methods current machines are moving away from the simple model of cache behavior modeled in this chapter. The additional architectural complexity allows the current generation of machines to continue to increase in performance through the use of higher clock rates. Without these features, the performance of machines over 300MHz would be limited by the time needed to process cache misses.

Because processor speeds continue to increase faster than memory system speeds, the effects of memory system performance will soon be felt by nearly all processors. One of the benefits of using GaAs DCFL logic is that high clock rates can be achieved. This required the Aurora processors to cope with memory system delays of hundreds of clock cycles. Such long memory delays have a fundamental impact on processor microarchitecture. The remaining chapters of this dissertation study the impact of GaAs technology and high clock rate on processor microarchitecture.

CHAPTER 3

Gallium Arsenide Microprocessor Design Studies

The goal of the University of Michigan Aurora project was to evaluate E/D MESFET Gallium Arsenide technology for digital VLSI applications. This chapter outlines the early work in the Michigan GaAs microprocessor project. A multi-phase development effort was planned, with the project divided into three main chip design phases. Each phase was intended to explore the design space at successively finer levels of detail and to build on the results of the previous phases. This chapter highlights some of the background work necessary to establish a strong foundation in Gallium Arsenide process and circuit fundamentals.

There were several critical questions to answer. First, was a microprocessor-size chip possible with acceptable yield? Second, what were the strengths and weaknesses of the GaAs process technology; what circuit design techniques could be used to achieve a dense, fast design? Third, could a performance goal of a 250 MHz clock frequency be met? Finally, did it make sense to build a 250 MHz chip, or would cache misses result in so much lost performance that the high clock speed would result in no performance improvement? These questions would be answered as the project progressed through three phases, corresponding to the design of the Aurora I, II and III processors. The design of the Aurora III processor is the subject of Chapters 4 through 7.

The phases were chosen to build from a process-oriented level to an architectural level. Phase one had three primary goals: to gain experience in GaAs circuit design, to develop an integrated set of CAD tools for high speed GaAs processor design, and to evaluate the yield of large GaAs circuits. The test vehicle for the first phase was a reduced-function CPU chip. The second phase was concerned with high-speed circuit design and cache control. This would require a second test CPU, designed for high clock frequency. The primary issues to be addressed were high-speed chip interfacing issues, signal integrity, clock gener-

ation and on-chip clock skew management, pipeline stalling, and cache control. The final phase was the design of the full CPU chip set, using the knowledge obtained in the first two phases. One challenge in an academic environment is the high turnover rate associated with graduate education. To gain experience quickly and apply it to the design of the next phase, a rapid design turnaround cycle was chosen as a major goal for all phases of the project.

3.1 Gallium Arsenide DCFL Logic

Since the invention of the Gallium Arsenide Metal Semiconductor Field Effect Transistor (GaAs MESFET), there has been continued progress in the technology, and chips with over 500,000 transistors are now possible. However, the density of GaAs continues to lag about 3-5 years behind that of state-of-the-art CMOS processes [Brown92].

Inherent technological weaknesses in GaAs account for much of this process lag. It is a very brittle material, limiting the maximum wafer size to 200 mm. Thermal gradients in the LEC crystal growth process used to produce the wafers results in higher crystal defect rates for GaAs than for silicon. Unlike silicon, GaAs has a poor native oxide. The material is also anisotropic: electrical behavior of the transistors differs depending on how they are oriented with respect to the crystal structure of the wafer.

Topology

Figure 3.1 shows a GaAs DCFL NOR gate, which uses a circuit topology similar to silicon NMOS logic. A depletion FET supplies current to a collection of parallel pull-down

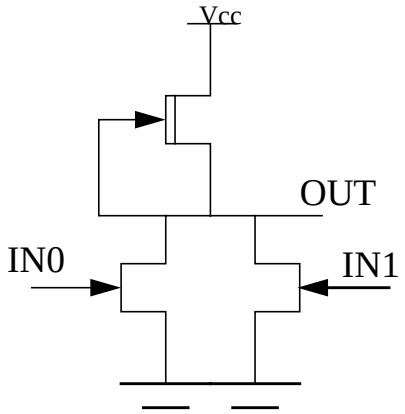

Figure 3.1 2-Input GaAs NOR Gate

devices to realize a NOR logic function. The depletion load supplies current to create the logic high voltage, and the enhancement devices pull the logic gate output low to create the logic low voltage. The depletion transistor threshold is usually about -0.6 Volts and the enhancement threshold is approximately 0.2 Volts.

The high source resistance of GaAs MESFETs limits the use of series enhancement transistors, which are needed to build NAND gates; the device sizings required to ensure proper noise margin make them so slow they have little use.

The small enhancement threshold requires a solid ground supply distribution network across the entire chip. This is provided by dedicating a full metal layer to the distribution of the ground supply, with a ground plane over the entire chip on the uppermost layer.

Leakage Current

MESFET devices have large subthreshold leakage currents between the source and drain terminals. At high temperatures, this leakage current can be several microamperes. Because the typical load device is sized to provide 100 microamperes, a large gate fan-in can severely affect the logic high noise margin of the gate. To control this problem, most gates are limited to a fan-in of 4.

Shottky Barrier Gate Diode

Perhaps the most distinguishing feature of GaAs E/D MESFET transistor is the lack of an insulating oxide between the gate and the channel. Instead of an oxide, the metal-semiconductor interface forms a Schottky Barrier Diode between the gate and channel. This diode clamps the logic high voltage of the inputs to 0.7 V, causing the gates to have a small logic swing: 0.1V for logic low to 0.7V for logic high. This small swing is responsible for some of the speed of GaAs circuits.

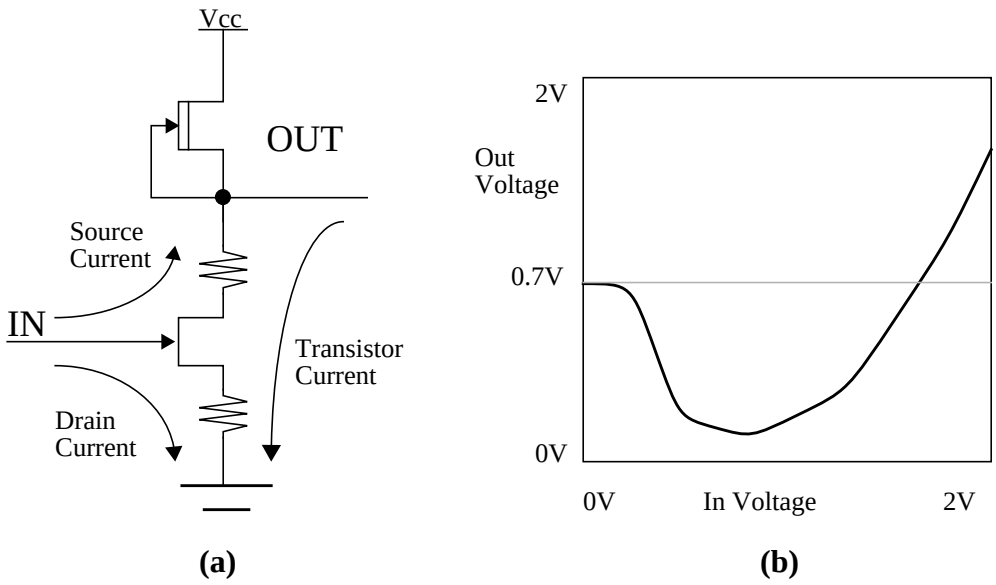

Figure 3.2 GaAs Inverter Transfer Function

Transfer Curve

The high source resistance and diode gate connections combine to create the unusual transfer curve for the DCFL gate, shown in Figure 3.2b.The normal operating range for the gate input is between 0 and 0.7 V. With the gate voltage at 0.7 V, the pulldown transistor is turned on and the output node is pulled down by the transistor current. As the gate voltage is raised above 0.7 V, the gate diode becomes forward biased and conducts current into the source and drain resistors. If the input voltage is too high, the gate diode becomes strongly forward biased and can conduct significant current. The diode current splits between the source and drain nodes. In extreme cases, the current from gate to drain can increase the output voltage, causing an invalid logic level or even a false logic ONE when a logic ZERO is desired. This condition can occur when a large buffer is used to achieve a short delay on a highly capacitive signal line.

Figure 3.3 shows what happens in this case. A large current is needed to charge and discharge the line quickly, but this current may be too large for the static current sinking capability of the gates connected to the line, possibly resulting in overdriving and incorrect logic values. The invalid logic values are shown as shaded regions in the figure. The small invalid regions result from the output passing through the indeterminate output range as the

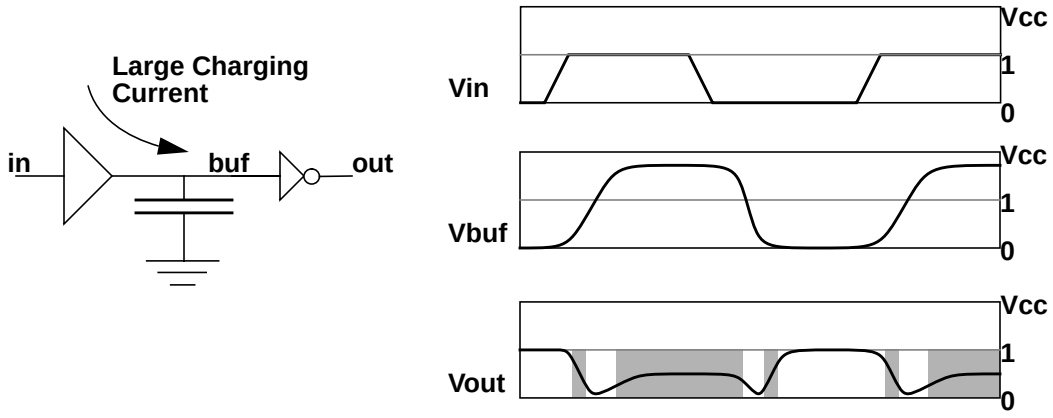

Figure 3.3 GaAs DCFL Buffer Overdriving

output passes from one valid logic state to the opposite logic state; these invalid periods are transient. The large invalid regions result from overdriving, and are static.

3.2 The Aurora I Processor

To gain experience as a design team, a multiple phase development plan was defined for the project, which included the development of two intermediate processors. Each test processor was designed using an advanced set of CAD tools, which allowed rapid design turnaround. This rapid design cycle enabled the design team to learn from the mistakes in the design of each processor before the next processor was begun. The Aurora I processor was fabricated in the Vitesse HGaAs II DCFL technology. HGaAs II is a 1.2 micron (1 micron L_{eff}) process with 3 metal layers. To ensure proper ground distribution, the uppermost metal layer was devoted to a full-chip ground plane layer, leaving 2 metal layers for signal routing.

The Aurora I chip was primitive, implementing a 28-instruction subset of the MIPS R2000 architecture [Brown93]. A diagram of the Aurora I pipeline is shown in Figure 3.4. The instructions implemented included word loads and stores, a full complement of jump and branch instructions, addition, subtraction, and logical operations. Unimplemented instructions included shifts, system calls, and sub-word memory operations.

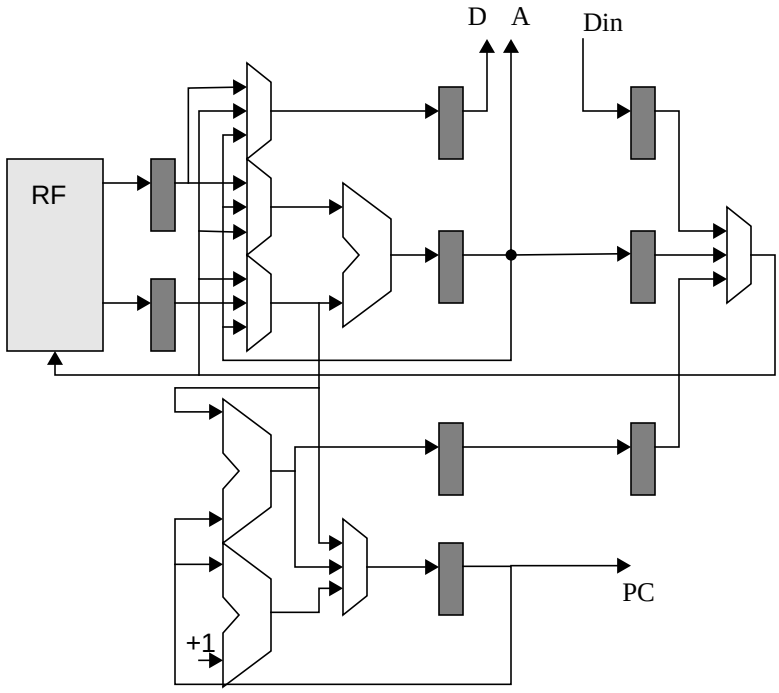

Figure 3.4 Aurora I Pipeline Diagram

A conservative design was adopted for the three-port register file needed to implement the MIPS R3000 architecture. A sense-amplifier-based register file was considered, which may have provided higher performance, but it was deemed too risky, in both performance and yield. Instead, a register file was built that used a multiplexor tree to read the data, and 4-bit latch cells for data storage. The register file design is shown in Figure 3.5.

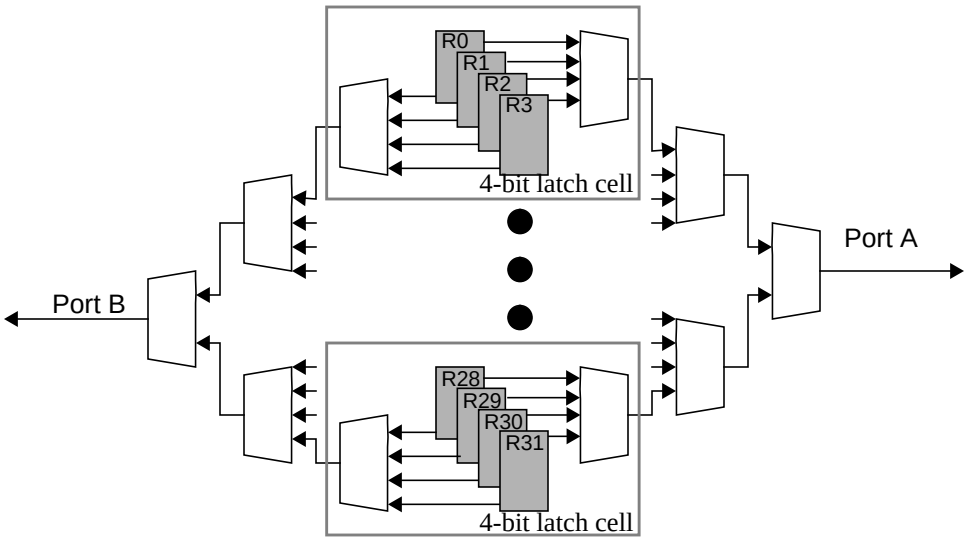

Figure 3.5 Aurora I Register File Organization

The chip was designed to be tested only on an IC tester, and had no provision for the control of external instruction or data caches. Because there was no support for caches, and the processor did not implement exceptions, there was no need to stall the pipeline. This greatly simplified the control logic. Figure 3.6 shows the organization of the control logic for the Aurora I processor. The instructions enter the chip at the left and proceed down a four-stage pipeline. The full instruction is passed to each successive stage, and is locally decoded to generate the control signals needed at each stage of the datapath. This control logic is in series with the datapath logic, possibly increasing the length of the logic paths. This control approach limited the clock frequency of Aurora I, but high clock frequency was not the major objective for this design.

3.2.1 CAD Tool Development

Traditional IC design methods based on schematic capture and gate level simulation were reaching their limits in the late 1980's as chips grew in size and performance. A primary goal of the first phase of the project was to determine the CAD tool requirements and build an initial CAD system. Two tools were chosen as the foundation of the CAD system for this project: the Verilog simulation language from Cadence and the Epoch layout generation system from Cascade Design Automation for translating these Verilog descriptions

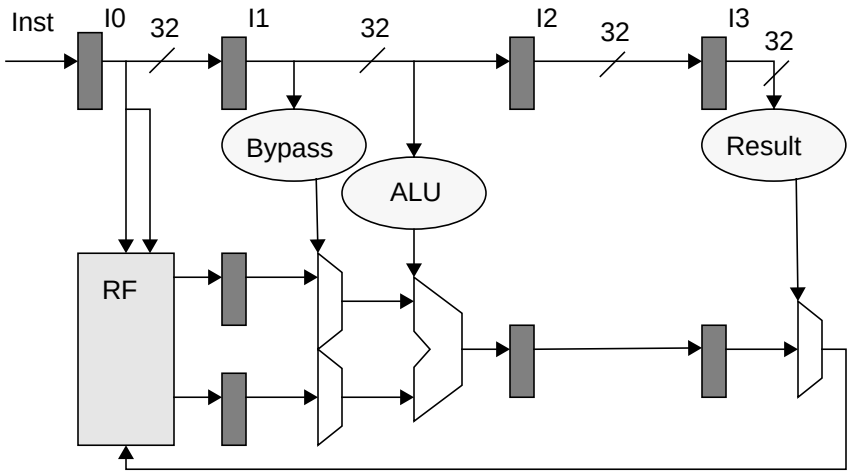

Figure 3.6 Aurora I Control Design

into physical layout.

The low level library cells used to build our chips were developed by our group using the Cascade Compiler Development System (CDS). The chip was laid out using a mixture of logic layout styles. Regular bus oriented logic structures were grouped into datapath blocks, and were automatically placed and routed using algorithms that optimize over-cell routing. The control logic was built from standard cells, and the I/O pads were hand generated using full custom layout.

3.2.2 Aurora I Test Results

The chip was submitted via MOSIS for fabrication by Vitesse in July 1991. Twentyfour prototype chips were packaged and delivered for testing. The CPU was found to have one human design error that disabled the Program Counter output bus. The error was caused by an invalid combination of normal DCFL and source-follower buffers, and was detected shortly after submitting the design for fabrication, but too late to be corrected. An I/O bond-

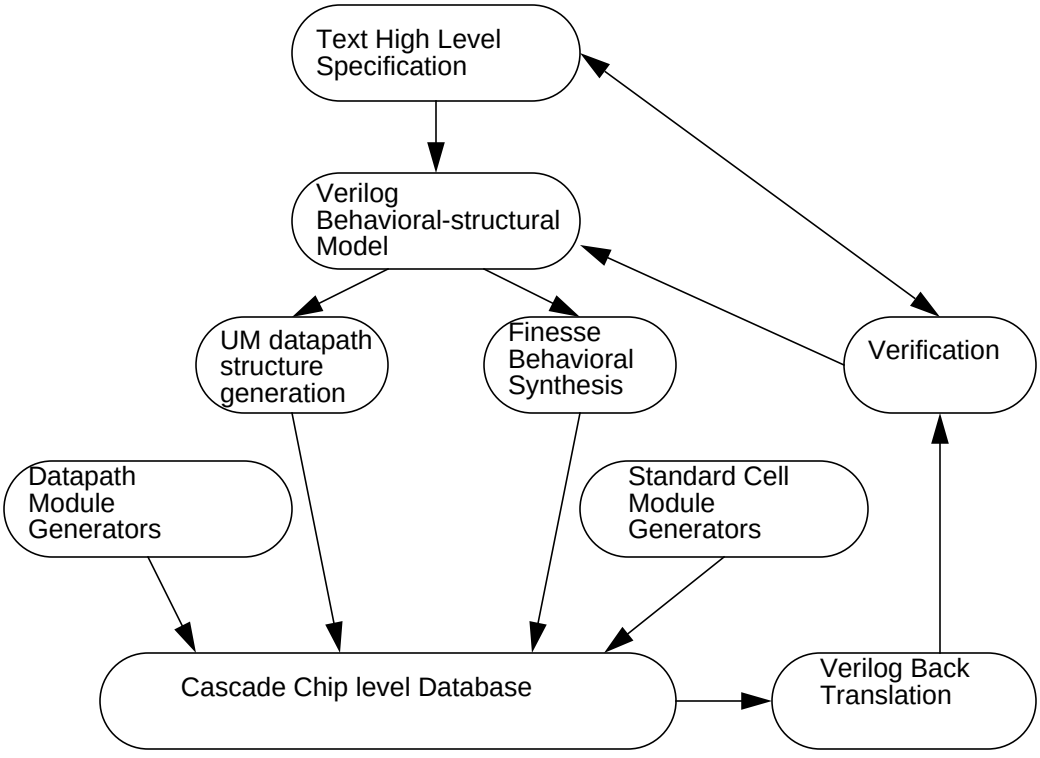

Figure 3.7 Aurora I CAD System Flow

ing problem forced all tests to be run using the internal scan chains.

Extensive functional testing was performed on the register file to evaluate the process yield characteristics. These results are summarized in Table 3.1. Four chips, or 16.7%, contained fully functional register files. Using asymmetrical clocks, the four fully functional register files had worst case access times of 6.4 to 6.7 ns. These four chips, plus two others, had fully functional ALUs. Propagation delays of 6.2 to 8.0 ns were measured in the ALUs, with an average of 7.25 ns. The fastest chip passed both ALU and register file tests with a 7.3 ns clock cycle, for a 137 MHz operating frequency. These two blocks are not guaranteed to contain the chip critical path and so the maximum chip frequency may be somewhat lower.

Although there were some design errors in the chip, the fundamental question of whether it was possible to build GaAs microprocessors was answered positively. A primary finding resulted from the design error that disabled the instruction address bus. This error was caused by connecting source follower logic and normal DCFL logic to the same electrical node. The DCFL logic prevents the source follower logic from achieving proper logic levels. Since then, only compatible logic families have been used within a chip. More advanced CAD tools could also be used to solve this problem.

In the Aurora I design, a primitive circuit method was used to solve the overdriving problem in large buffers. A diode to ground was added to the output of each buffer cell, providing a path for the excess current after the gate output voltage reaches the logic high

functional	number	percentage	probable cause
fully functional	4	16.7%	
single bit failures	5	20.8%	random defects
entire register stuck at 0 or 1	2	8.3%	Read or Write decode circuitry failure
same bit failure in multiple registers	7	29.2%	read out multiplexer failure
failure of all registers	6	25%	clock or power distribution

Table 3.1 Aurora I RF Yield Analysis

value. This method is not an effective solution for large chips because the gate output current remains high, resulting in high power dissipation. If used for a processor with hundreds of thousands of transistors, the diode clamping method would dissipate hundreds of Watts.

3.3 The Aurora II Processor

The Aurora I processor showed that large GaAs chips were possible, but did not give much information about the performance achievable in the technology because of the lack of cache support and timing optimization. The primary goal of the Aurora II processor design was to explore issues in high-frequency processor design with on-chip support for cache memories.

The Aurora II processor was fabricated in Vitesse HGaAs III DCFL technology. HGaAs III is a 1.0 micron (0.8 micron L_{eff}) process with four metal layers. To ensure proper ground distribution, the uppermost metal layer was again devoted to a full-chip ground plane layer, leaving three metal layers for signal routing.

The Aurora II register file was of the same design as that used in the Aurora I processor. To increase the circuit density, the first level of readout multiplexors was integrated with four memory latches to form a four-bit-primitive register-file cell.

Figure 3.8 shows the Aurora II pipeline. The primary design goal of the second test processor was to support a multilevel cache hierarchy. To simplify the CPU-Cache interface, all signals required for cache control are generated by the CPU, eliminating any external handshaking logic. The CPU was designed to directly control two 4K-byte instruction and data caches. Each cache would be made from a pair of 1K by 32 SRAMS, one holding the data and the other holding the tag. Both caches would have a one-word line size, and the data cache would operate in write-through mode. A single port version of the register file

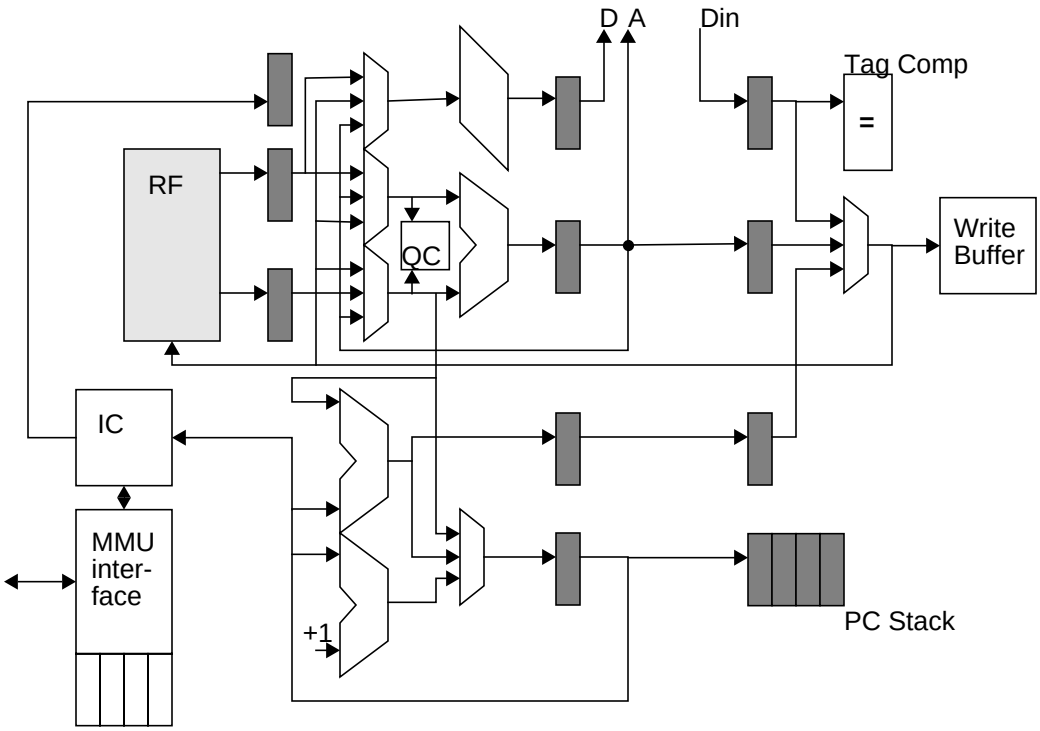

Figure 3.8 Aurora II Pipeline Diagram

was used to construct a small on-chip instruction cache for high speed testing.

Because the goal for the Aurora II chip was a high clock rate, as much control logic as possible was removed from the critical paths of the chip. Figure 3.9 shows the organization of the control logic for the Aurora II processor. Unlike the Aurora I chip, all the control sig-

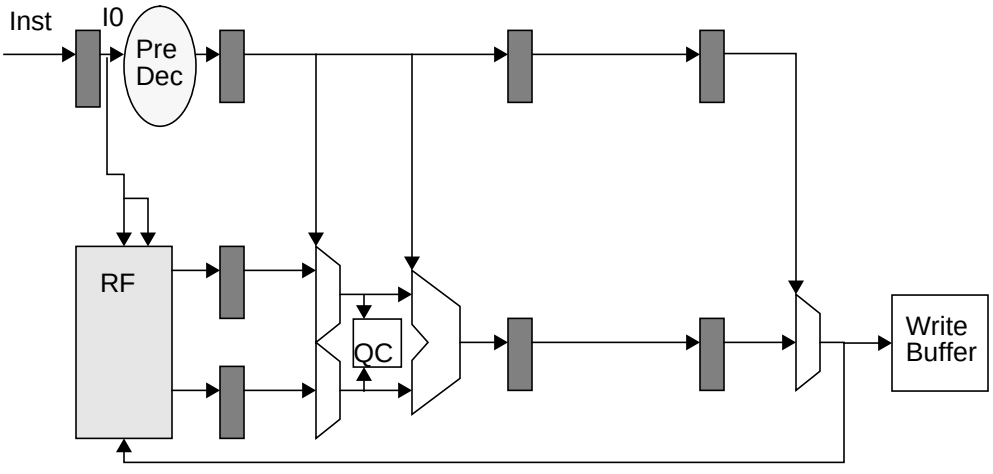

Figure 3.9 Aurora II Control Logic Diagram

nals for the Aurora II chip were generated early in the pipeline, so the 32-bit instruction did not have to be propagated the entire length of the pipeline. Since the control signals came directly from latch outputs, few control signals were on any of the critical paths.

Cache-miss stalls and exceptions cause many design difficulties in a high clock-rate processor. Two primary methods are used in pipelined machines to deal with stalls: pipeline freeze and pipeline restart. As its name indicates, pipeline freeze stops the pipe completely, and releases the pipe when the condition causing the stall has been removed. While the pipe is frozen, each pipe stage maintains all instructions and data constant, ready to flow down the pipe when the stall condition has been resolved. Pipeline freeze was used in the Multititan processor [Jouppi89].

There are two ways to freeze the pipe; the data may be recirculated using multiplexors, or the clock to the latch can be gated. Clock gating has been viewed with suspicion by designers, due to difficulties ensuring correct functionality throughout the operating range of the chip. However, with the recent emphasis on low power design, this method is becoming more common, reducing the average number of active nodes, and thus minimizing power dissipation. More work is needed to adequately handle gated clock designs in timing analysis and power estimation tools; this remains an active research area.

The other stall method, pipeline restart, records the address of the offending instruction, and restarts the offending instruction from the beginning of the pipe when the stall condition has been removed.

In the Aurora II design, both stalling and exceptions were implemented using an instruction restart design. When a cache miss was detected in either the instruction or data streams, the address was saved, and when the miss had been handled, the instruction was re-executed. This scheme results in more execution time overhead, and may not be the proper choice for a commercial machine. Instruction restart was used to minimize the number of pipeline stalling conditions. Stalling logic is often on a processor’s critical path.

An asynchronous interface to the MMU was designed, allowing the CPU chip to operate at its maximum clock rate. The Aurora II processor has three cache modes: Buffer, For-

ceMiss, and Cached. In buffer mode, the internal cache tag checking is inhibited, and the CPU executes the instructions presented at the Instruction input bus, but because cache misses are ignored, no MMU bus transactions occur. Cache mode and ForceMiss mode both use the MMU interface. In ForceMiss mode, all cache accesses are treated as misses, and the required instructions and data are fetched over the MMU interface. In cache mode, the data are fetched over the MMU interface and the data and tag values are updated in the external caches.

A four-entry write buffer minimizes in Aurora II the effects of write-back traffic on CPU performance. To facilitate high-speed testing, a small 32-word instruction cache is included on-chip. Test programs can be loaded into the on-chip cache at low speed using the asynchronous MMU interface and then executed at full-chip speed.

The CPU and MMU communicate via a set of four interface registers located on the CPU chip. The registers include address and data registers for both the instruction and data streams, allowing instruction misses and data misses to be processed in parallel. When a cache miss is detected, the CPU writes the miss address into the appropriate MMU interface register, and signals the MMU with the type of request. Three different requests are supported: I-cache miss, D-cache miss and Data-writeback request.

The interface between the CPU and MMU is asynchronous, allowing each to operate at maximum speed. After the CPU writes the requested address and signals the MMU, the MMU reads the address register, fetches the requested data from the secondary memory system, and returns the instructions or data to the appropriate data register on the CPU. The CPU detects that the requested data has been supplied and restarts the CPU pipeline, simultaneously writing the data into the off-chip cache using the normal execution pipeline. Direct bypass paths from the MMU interface to the data and instruction inputs allow the CPU to run programs at full CPU clock speed with no caches attached, although the slower speed of the MMU interface will limit throughput.

3.3.1 Timing and Optimization

To minimize problems with clock skew, a two phase clocking scheme, as shown in Figure 3.10, was adopted for the Aurora II processor. A six stage clock buffering network distributed the clocks to the destination latches with about 400 ps of clock skew.

Kayssi developed a set of timing macromodels for the GaAs DCFL gates [Kayssi93]. The macromodels accurately predict the effects of varying input rise times, output capacitance and output diode load on the propagation delay and output slew rate of the gates. The macromodels were integrated into the Cascade Design Automation timing analyzer, and were used extensively in the timing optimization of the chip.

Critical path optimization is a time-consuming and error-prone task. Much of the task consists of two primary operations: logic tree height reduction and latch retiming. Both of these tasks could be automated, but as yet, have not been added to our design tool suite. Manual retiming and hand logic design were used to improve the quality of the Aurora II circuits.

Figure 3.11 shows the improvement in maximum clock phase delay over five iteration of retiming and manual logic design. The maximum delay between latches was reduced by nearly half, from 8.3 ns to 4.4 ns. The two final iterations show that it is sometimes necessary to make the result worse to reach a more optimal global solution. Automatic tools need some kind of stochastic search such as simulated annealing to automate this process.

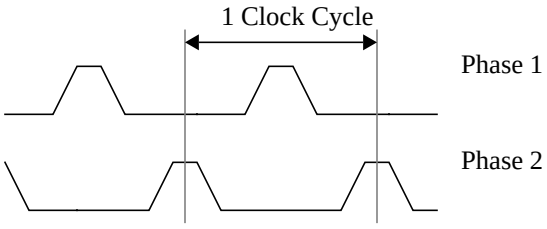

Figure 3.10 Two Phase Clock Diagram

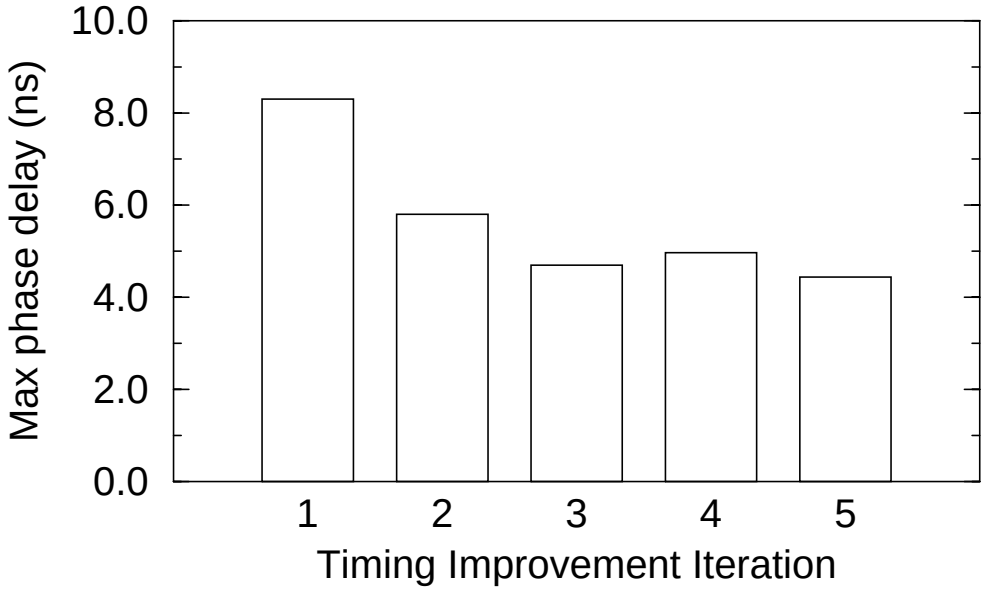

Figure 3.11 Manual Cycle Time Improvements

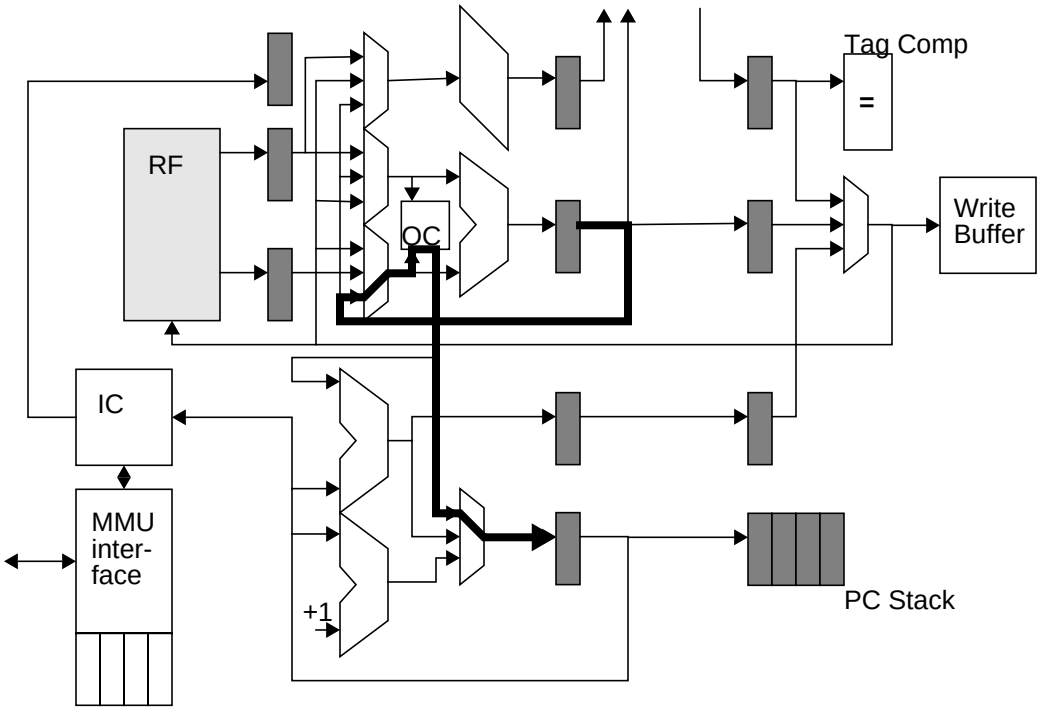

Figure 3.12 Aurora II Critical Path

Figure 3.12 shows the final critical path in Aurora II. The path begins at the output of the ALU and passes through the bypass network to the quick-compare logic. From the

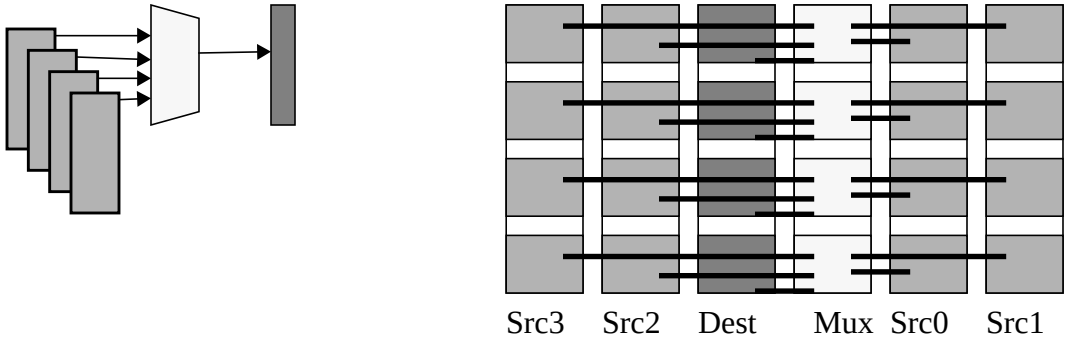

Figure 3.13 Column Based Datapath Example

quick-compare logic the path goes through a few gates of random logic that control the next address multiplexor for the program counter. The final delays achieved were 4.4 ns for phase 2 and 4.3 ns for phase 1. These delays include a total of about 2.5 ns of clock tree buffering delay, so the predicted minimum clock period is about 6.2 ns, for a maximum predicted operating frequency of 161 MHz.

3.3.2 Layout Optimization

The Cascade Design Automation physical layout tools provide layout generators for several layout styles. A unique feature of the Cascade toolset is the Datapath compiler. A datapath is a block of bus-oriented logic consisting of Register Transfer Level (RTL) components. The RTL components, such as latches, adders and multiplexors form multi-bit stacks of logic called datapath columns. Each column implements an N-bit wide RTL logic function. Multiple columns are grouped together to form datapath logic blocks.

A weakness of the production Cascade tools at the time of the Aurora II design was the poor quality of the datapath column placement. A small datapath example is shown in Figure 3.13. This datapath consists of six column elements, each of which is four bits wide. The datapath placement task orders the columns such that the net lengths between the columns are minimized, and the interconnect signals are routed over the cells.

The original Cascade placement tool often did a poor job of ordering the columns, resulting in large net lengths and larger layouts. Based on work done by Cai [Cai90], a placement program that uses a branch and bound algorithm to optimally determine the

Datapath	Cascade Placement (meters)	U of M Placement (meters)
pipe0	2.49	2.28
reorder	2.01	1.32
pcdpath	4.33	2.23

Table 3.2 Optimal Datapath Placement Results

minimum-cost column ordering was written at the University of Michigan. Table 3.2 shows the improvement given by the new algorithm on three datapath examples from the Aurora III design; on average, the datapath netlengths were reduced by 31%. The datapath block areas were reduced by an average of 16%. Because the datapath blocks are critical elements of microprocessor designs, these savings directly impact the resulting chip performance. The datapath placement algorithms developed for this project have been incorporated into the commercial tools offered by Cascade Design Automation.

Random logic used to generate buffered clocks and multiplexor select lines accounted for much of the area in the Aurora I processor. Nearly every datapath multiplexor and latch requires a select line buffer or a clock buffer. In the Aurora I, these gates were implemented as standard cells and placed in groups near the target datapath. Although the gates were near their destinations, a substantial amount of routing was often needed to make the connections. If these buffers were integrated into the datapath, the area of the chip would be significantly reduced, and the interconnect length predictability improved. Figure 3.14 shows the solution that was implemented. The local decode and buffer logic was grouped into an additional datapath cell and placed above the datapath column. This increased the number of cells that had to be designed, but resulted in significant area and route-length improvements.

In addition to the column drivers, the cell set was redesigned for Aurora II to take advantage of the third metal routing layer. This allowed the control and clock lines to be routed vertically over the top of the datapath cells. Data busses were routed horizontally between cells using the second metal layer, and also routed on top of the cells. This two-

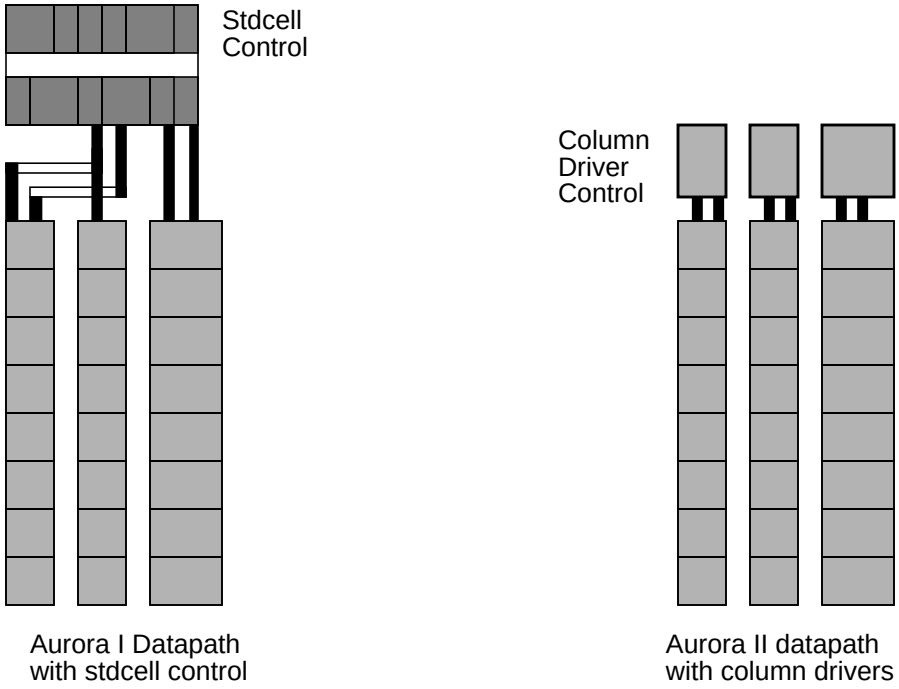

Figure 3.14 Datapath Column Drivers

dimensional overcell routing increased the routing layer utilization, and improved density by nearly 100% over the Aurora I design.

3.3.3 I/O Pad Design

The Aurora II processor was designed to use either commercial ECL SRAMs or custom GaAs SRAMs for external primary caches. ECL logic uses a 1.2V logic swing; the GaAs SRAMs were designed to use native GaAs levels with a 0.7V swing. An output pad capable of interfacing to either level was required.

The output pad uses gated feedback to control the logic high output level. Figure 3.15 is a schematic of the output pad. The first three logic stages form a GaAs superbuffer that drives the output pullup node. MESFET Mx is configured as a diode, dropping the pad output high voltage one diode drop below the 2V VCC supply.

The GAAS signal controls whether the pad operates at ECL or GaAs logic levels. With the GAAS signal low, the output feedback is disabled and the pad operates as an ECL output pad. In this mode a logic low on the Input pad causes a low value on the Pullup node,

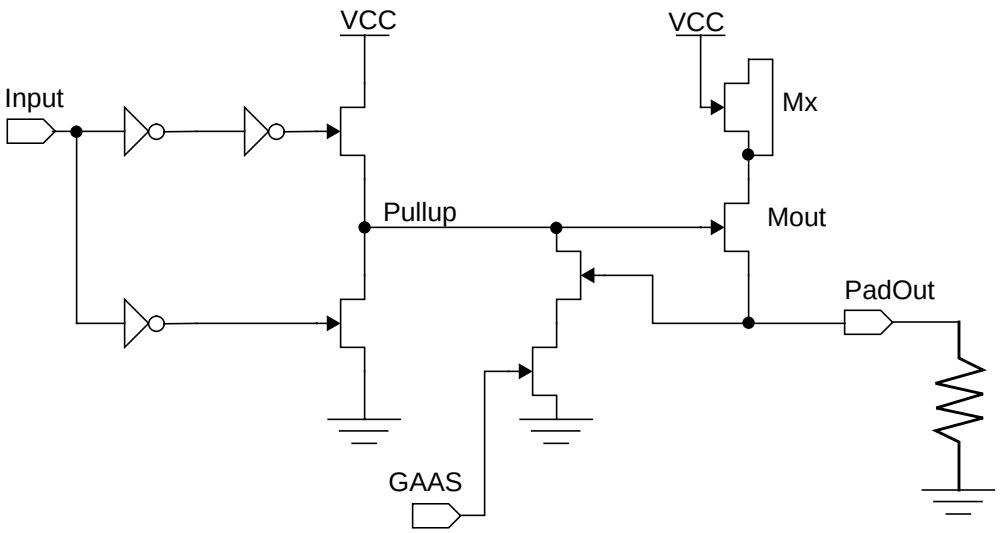

Figure 3.15 GaAs Electrically Programmable Output Pad

turning off the output pullup transistor, Mout. With the output pullup off, the external termination resistor pulls the pad output to 0V. A logic high value on the Input pad brings the Pullup node high, turning on Mout. The PadOut voltage is pulled up to one diode drop below VCC, giving the 1.2V ECL logic swing.

The pad operation in GAAS mode is more complicated. A logic high on the GAAS signal enables the feedback FET connected to the PadOut signal. A logic low on the Input gives a logic low on the Pullup node, and the pad is turned off as in ECL mode. When the input switches to a logic ONE, the output begins at 0V and the feedback device is off. The superbuffer switches fully on, turning on Mout. The PadOut voltage begins to rise, turning on the feedback FET. The feedback FET drains current from the superbuffer output, reducing the voltage on the Pullup node. As the output voltage increases, more current is drained from the superbuffer, further reducing the Pullup voltage. Reducing the Pullup voltage reduces the output current, limiting the maximum output voltage to about 0.8V. The feedback scheme used to clamp the voltage is similar to that used by Fulkerson for Feedback FET Logic [Fulkerson91].

3.4 Aurora II Results

The Aurora II processor had several design errors. These errors had three primary caus-

es: inexperience of the designers, inadequate design tools, and insufficient testing. Two fabrication runs were required to obtain parts that could be adequately tested.

3.4.1 Error Summary

The initial design was submitted for fabrication in July, 1992. After the design was submitted, an error was discovered in an MMU interface state machine. The error was caused by a missing state transition arc in the CPU stall state machine. The error could have been detected earlier by varying the timing parameters in the MMU model.

Forty packaged chips were received from MOSIS in late September of 1992. Initial testing showed that the clock distribution logic was functional, and some intermittent functionality was observed over the MMU bus. Power consumption was too high, and analysis identified a short circuit between the power and ground planes of the chip in one corner of the die. The short was caused by use of an “anti-Metal-3” layer in the Vitesse I/O pads. Vitesse uses this layer to cut out sections of metal-3 routing for connecting ground plugs to the lower metal layers. MOSIS does not support this layer, so the metal 3 masked by the anti-metal-layer was removed by hand from the pads. The anti-metal-layer itself was not removed however, and a subsequent hand edit overlaid a large metal 3 VCC box on the anti-layer boxes. Our local verification software recognized the anti-metal-layer information and showed no error, but the MOSIS software ignored the anti-metal-layer information, creating a solid metal 3 layer over a large section of the chip. Since the added metal 3 supplied power, and the anti-layer was used to isolate ground, the power and ground planes were shorted together.

Testing was continued with the short, and the best three chips were identified. The short circuit was removed from those chips by ion-beam milling. There were 11 shorts, which required about three hours of milling per chip to correct. Two of the ion-milled chips showed good functionality. Simple tests of each instruction verified that all the instructions worked. The ion beam milling damaged some of the configuration pins of the chip, so all testing had to be performed in buffer mode, with the caches and MMU interface disabled. The maximum frequency the entire chip would operate was 100 MHz, which was only

functional	number	percentage	probable cause
fully functional	2	2.5%	
single bit failures	1	1.25%	random defects
entire register stuck at 0 or 1	3	3.75%	Read or Write decode circuitry failure
same bit failure in multiple registers	4	5.0%	read out multiplexer failure
failure of all registers or MMU interface	69	86%	MMU interface failure, clock or power distribution, datapath failure

Table 3.3 Aurora II RF Yield Analysis

about half of the expected operating speed. The program counter datapath operated at up to 200MHz, however. The speed-limiting path was isolated to the instruction decode logic. A clocking phase error had been introduced late in the timing optimization of the design. The instruction decode logic for both phases used inputs of the wrong phase. To obtain the correctly decoded instruction, it was necessary to set the active interval of each phase to the flow-through delay of the instruction decode logic. This extended the minimum clock cycle by a factor of two.

The logic affected by the clock phase error was evaluated, and the best approach to fixing the error was to hand edit the layout. The second fabrication resulted in chips that could be more easily tested. Further testing of these chips revealed an additional class of bugs in the instruction decode logic, caused by a design error in the specification of some of the immediate instructions. The effect of this bug is to sometimes interpret the low order bit of an instruction as an immediate instruction when the instruction is of a different type. This causes the instruction to generate two simultaneous and conflicting decodings.

Register files and scan chains for the Aurora II processor were tested for DC functionality [Powell94]. The results for the register files are summarized in Table 3.3. The results indicate severe problems with process yield. Studies performed by Chandna attributed much of the yield loss to a lack of process tolerance in the high-current buffer design that we had invented [Chandna94]. This buffer was a variation of a Honeywell superbuffer de-

sign [Fulkerson91]. The circuit was modified assuming a much smaller range of process variation than achieved in fabrication. This circuit has been eliminated from our current cell set.

No effort was made to equalize the amount of logic in each clock phase. This resulted in a critical path with 38 gates, 24 of which were in one phase. Uneven logic level depth requires an asymmetrical clock. This clock can be difficult to generate and distribute at high clock frequencies.

3.4.2 Clocking Issues

Clock distribution is a challenge in any high speed processor. A six-level clock buffer tree was used in Aurora II to distribute the clock to the control logic and each of the datapaths. The column drivers in the datapaths provided good local buffering of the clock. The maximum delay of the clock distribution network was 2.5 ns, one half of the 200MHz clock period. Such a large clock distribution delay makes it difficult to interface multiple chips together, because the off-chip delay time can vary widely from chip to chip due to process variation. By synchronizing the internal clock to an external signal, a phase locked loop can remove the uncertainty about when outputs from different chips are valid.

3.4.3 Exception Overhead

The Aurora II design added some essential functionality over the Aurora I design, mainly shift instructions, cache support and exceptions. The associated increase in implementation complexity was enormous. The Aurora I processor required about 944 gates and 128 latches to implement the control logic. Of the 944 gates, 257 were high-current buffers needed to drive the datapath control lines. The Aurora II design, in comparison, required 1256 gates and 277 latches in the control logic.

Much of this added complexity came from the support of stalling, exceptions and the asynchronous MMU interface. To support an external data cache, a full program counter stack was added to the Aurora II chip so the address of the data-cache-miss instruction

Logic Module	CMOS gate count	GaAs gate Count	CMOS Average levels	GaAs Average Levels
wc_w0	162	302	8.53	9.87
wc_fpbusy	107	271	6.77	9.00
eupredec	159	254	7.09	8.17
issue_cntl	88	150	7.25	7.11

Table 3.4 Logic Synthesis Comparison

could be saved to restart the instruction.

The MMU interface added a set of interface registers and a write buffer; these also needed control logic that increased the complexity.

3.4.4 GaAs Technical Difficulties

Many process characteristics combine to make GaAs DCFL circuit design more challenging than design in other circuit families. The lack of any series transistor structures removes much of the flexibility and creativity from circuit design, and forces the designer to use only NOR gates for building circuits. This restriction has two primary detrimental effects on logic blocks in GaAs. First, significantly more logic levels are required to implement a function in DCFL than in CMOS. A direct result of the increase in the logic levels is an increase in both the number of gates required and the amount of routing needed to connect the gates.

Table 3.4 shows logic depth results for some of the major control blocks in the Aurora III design. When implemented in DCFL, these blocks require an average of 15% more gate levels than the same function realized in CMOS. This reduces the benefit of fast gate switching speeds. Because the technology mapping in GaAs is constrained to use only NOR gates, a significant increase in the number of gates results. The GaAs implementation requires 91% more gates on average than the CMOS version. This increase in gate count causes an increase in the area required to lay out a logic function, which also reduces performance.

The Cascade logic synthesis tools optimized logic for area only; there was no option for optimizing performance. Any control logic in critical paths was therefore designed by hand. In the Aurora II chip, the hand-designed gates accounted for about 40% of the total number of random logic gates. With hand design, it was often possible to reduce both the gate count and the number of logic levels by close to 50%.

Due to the anisotropic nature of the GaAs material, there is inherently more deviation from the ideal behavior of the circuit elements. This circuit variation limits performance, because conservative design margins must be maintained; in the worse case it can cause the circuits to fail.

The enhancement/depletion DCFL logic family requires that several performance trade-offs involving noise margins be made. The logic gate's output low level must be below the pulldown transistor threshold. A maximum logic low voltage of about one half V_t is typically chosen. To ensure that the output of a logic gate reaches a good logic low value, the pulldown transistor must be stronger than the pullup by a factor of 3 to 4. This causes the pullup delay to be longer than the pulldown delay.

Because the logic low must be near one half V_t to minimize leakage currents, and the minimum logic low voltage is affected by process variation and device sizing, the threshold of the enhancement pulldown transistor must be quite high. In the HGaAs III process, the threshold is about 0.22 V; this is nearly 1/3 of the logic swing. The proportionally large threshold voltage slows the gate switching time when compared to a complimentary gate [Pfiester85].

The noise margin for a logic family can be determined using the inverter transfer function shown in Figure 3.2. The transfer function is replicated about the Vin=Vout axis, resulting in a figure-eight-like graph shown in Figure 3.16. This graph shows the transfer function of seven different inverters simultaneously. The inverters use different combinations of the process corners: fast, slow and typical. The seven different combinations used are listed in the common two-letter abbreviation where the first letter represents the parameters for the depletion transistor and the second letter represents the parameters for the en-

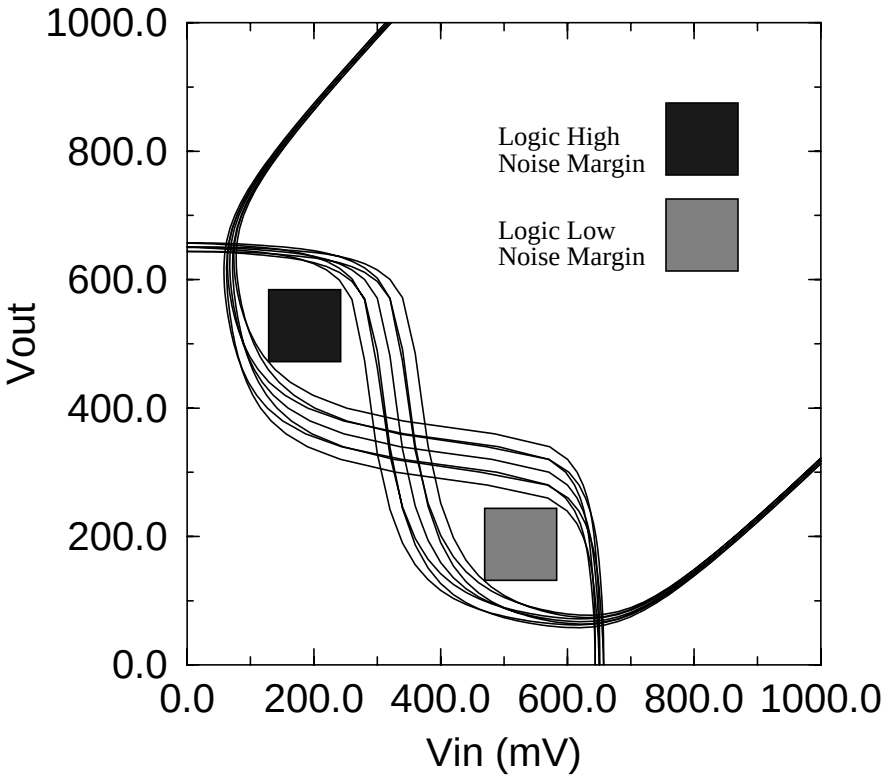

Figure 3.16 GaAs DCFL Noise Margin at 25 Degrees C

hancement transistor. The different inverters shown in the figure use the seven combinations: FF, TT, SS, FT, ST, TF and TS. The two other combinations, FS and SF, were not simulated because the performance of the enhancement and depletion models tend to track each other, and getting either of the two extremes in performance is unlikely.

Figure 3.16 shows the noise margin transfer functions for the DCFL inverters at 25 degrees C. The high and low noise margins can be determined by the maximum size square that fits in the open area of the figure eight. At 25 degrees C, the noise margin is about 120 mV. At 100 degrees C the noise margins degrade significantly as shown in Figure 3.17. The logic high and low noise margins have been reduced to about 60 mV.

Central to the success of any process for building microprocessors is the ability to support high speed static RAM. Because of the large process variation and low noise margins, this is very difficult in DCFL. Our group worked for three years on developing a robust RAM design for DCFL, and succeeded in meeting the goal of producing a RAM that works over a wide process variation, but with the sacrifice of some performance [Chandna94].

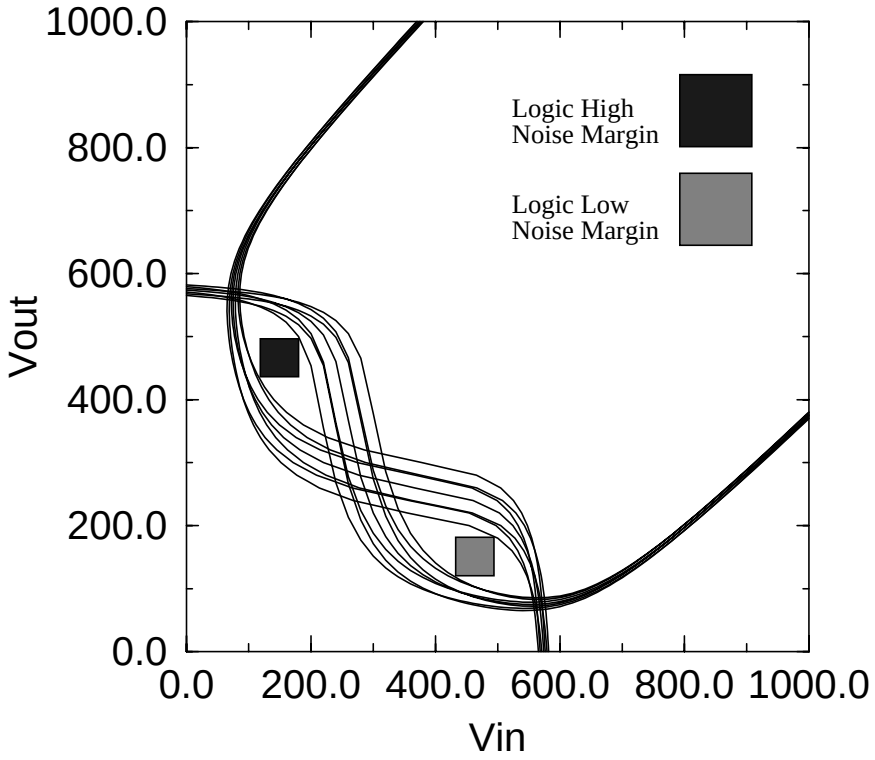

Figure 3.17 GaAs DCFL Noise Margin at 100 Degrees C

The RAM is also much less dense than current high speed SRAMS in CMOS technology.

The remaining chapters describe a microarchitecture, circuits and CAD tools developed to minimize the difficulties in using GaAs DCFL for processor design.

CHAPTER 4
Process Modeling Studies

4.1 SUSPENS system performance model

This chapter describes a performance model used to characterize different GaAs processes. The model is used to predict the area and performance of the Aurora III processor, and to establish initial frequency targets and logic depth targets. The importance of high density circuits and accurate estimation of chip area and delay parameters was demonstrated in the design studies of Chapter 3. Bakoglu has developed a model for computer system performance that can predict the system clock frequency, power and chip area using very high level information [Bakoglu87]. These models can be used to evaluate architectural decisions, and to see what limits are imposed by the processing technology.

The Bakoglu model needs few inputs to develop a chip or system estimate. The inputs and outputs are easily summarized on a single page, as shown in Table 4.1. The relationship

Parameter		Description
Inputs	f_{ld}	Critical path length in gates
	R_{tr}	Transistor on resistance
	n_w p_w e_w R_{int} C_{int}	Routing parameters
	f_g	Average fanout of gates
	N_g	Number of gates in design
Outputs	$\overline{R}$	Estimated average interconnect length in terms of gate pitch, Rent's Constant
	l_{av}	Estimated average interconnect length in microns
	d_g	Estimated gate pitch in microns
	T_g	Estimated average gate switching speed
	Dc	Estimated chip dimension in microns
	Pc	Estimated chip power
	f_c	Estimated chip operating frequency

Table 4.1 Performance Estimation Parameters

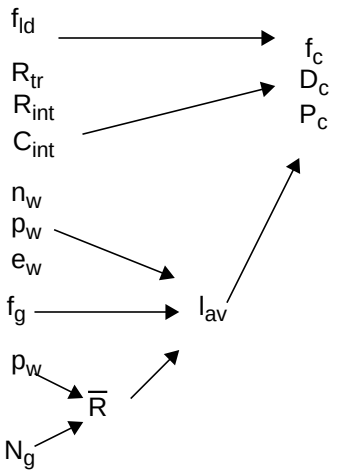

Figure 4.1 Parameter Flow Diagram

between the inputs and outputs is summarized graphically in Figure 4.1. Starting with inputs such as the number of logic gates in the design and the number of routing layers, the model derives an estimate for the area, power dissipation and maximum operating frequency for the chip. Varying the parameters of the model allows quick estimates to be made of the critical architectural and process features. For example, the effect on clock frequency of adding an interconnect level can be easily estimated.

4.2 GaAs SUSPENS Model

The two critical parameters in the SUSPENS model are wiring pitch p_w, and total gate count N_g. Depending on the wiring pitch, gate density, and circuit type, the size of the design will be dominated either by the area of the gates or the area of the wire connecting the gates.

The effective routing pitch for a process is computed from the actual routing pitch for each layer, p_w, the routing efficiency for each layer, e_w, and the total number of layers, n_w. Because the rules for routing pitch are very closely guarded, only the effective pitch number and the total number of routing layers for each process will be given for current processes. However, the usage of the model can be demonstrated using an obsolete GaAs process.

layer	contacted pitch, microns	utilization in cells	utilization in routing	effective pitch, microns
metal 1	4.3	50%	50%	8.6
metal 2	5.0	70%	30%	10.0
composite				9.25

Table 4.2 HGaAs II Effective Routing Pitch

The detailed rules for Vitesse HGaAs II are shown in Table 4.2. The effective pitch for each layer is determined by the actual routing pitch for the layer and the utilization of the layer. The layer utilization has two components, cell and area utilization. The utilization rates are determined empirically through analysis of the layer usage in cells and in routing channels. The chip is assumed to have equal areas occupied by cells and routing, so the global utilization is the average of the cell utilization and the routing utilization.

In a three-layer process like HGaAs II, in which one layer is dedicated to the ground plane, much of the chip area is used to distribute the positive power supply. This accounts for the low utilization of the metal 1 in the cells; though the cells are fully occupied with metal 1, much of this routing is used for power distribution.

Additional metal layers increase circuit density. The amount of increase is determined using a formula that is analogous to that used to measure parallel resistances. In the HGaAs II process, there are two routing layers, so the effective routing pitch p_{eff} is found by:

$$p_{eff} = \frac{N}{\sum_{N} \frac{1}{p_n}} \tag{13}$$

$$p_{eff} = \frac{2}{\frac{1}{p_1} + \frac{1}{p_2}} = \frac{2}{\frac{1}{8.6} + \frac{1}{10.0}} = 9.25 \quad : \tag{14}$$

layer	contacted pitch, microns	utilization in cells	utilization in routing	effective pitch, microns
metal 1	4.0	100%	80%	4.44
metal 2	4.0	70%	40%	7.27
metal 3	7.8	100%	50%	10.4
composite				6.53

Table 4.3 HGaAs III Effective Routing Pitch

The 3-layer metal routing pitch can be determined like the 2-layer metal pitch. The HGaAs III numbers were derived from information publicly available through MOSIS. Adding the third routing layer increases the utilization of both metal 1 and the highest routing layer because the area of the cell layouts is reduced, and there is less Vcc routing overhead.

Effective pitches for a variety of 3-metal routing layer processes are listed in Table 4.4. Because of the sensitive nature of process technology, the actual pitches are proprietary information and cannot be divulged, so only the effective pitch for the process is given. More recent CMOS processes have metallization rules based on tungsten via plugs. These processes are better planarized, and allow higher routing densities. For comparison to the GaAs processes, two high performance CMOS processes are listed. DEC CMOS4 is a 0.75 micron 3-layer-metal process with two high density layers and one high current layer used for power and clocks [Zetterlund92]. The wide spacing required for the high current metal

process	effective pitch, microns
Vitesse HGaAs III	6.53
Vitesse HGaAs IV	4.07
Cray GaAs	4.79
Motorola GaAs	3.27
DEC CMOS4	4.0
Intel 0.6 micron CMOS	2.1

Table 4.4 3 Metal Effective Routing Pitch

in the CMOS process increases the effective interconnect spacing, but this process is still competitive. The current generation Intel process is denser by far than any of the other processes listed [Schutz94].

A static technique is used to calculate the transistor ON resistance. The circuit used is shown in Figure 4.2. The current drawn by the transistor is proportional to the transistor ON resistance. Using this model, the R_{tr} parameter can be easily determined. Resistance values of 10K to 15K ohms per micron of gate width were typical for the GaAs processes listed.

In the mid 1960's, IBM performed several studies estimating the number of I/O pins needed to interconnect logic functions of different sizes. This information was used to determine the optimal density for each level of packaging hierarchy in a mainframe computer. It was determined that the number of I/O pins needed for a function grows at a slower rate than the number of logic gates needed for the function.

Empirical data showed that the best function for modeling the I/O requirement for a given logic design was exponential:

$$Num_{IO} = k\,(N_g)^p \tag{15}$$

For microprocessor designs, p is typically 0.4 to 0.6, and k is 1.0 to 1.5. The best fit for our data was p=0.45 and k = 1.0. Using this rule, the average interconnect wire length l_{av} can be determined using hierarchical decomposition.

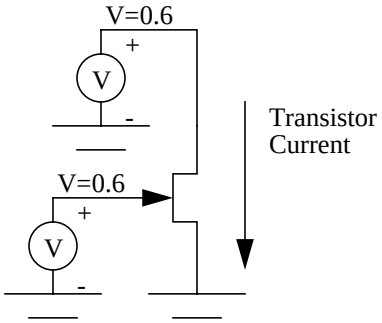

Figure 4.2 Transistor Resistance Calculation

4.2.1 Bakoglu Equations

The full derivation of these equations is given in *Circuits, Interconnections, and Packaging for VLSI*, Chapter 9 [Bakoglu90]. The computation of chip area and clock frequency is performed stepwise using the following procedure.

STEP 1: Calculate Rent's Constant, $\overline{R}$, the average net length in units of average gate dimensions using Rent's Rule.

$$\bar{R} = \frac{2}{9}\left(7\frac{(N_g)^{p-0.5}-1}{4^{p-0.5}-1} - \frac{1-(N_g)^{p-1.5}}{1-4^{p-1.5}}\right)\frac{1-4^{p-1}}{1-(N_g)^{p-1}} \tag{16}$$

STEP 2: Calculate the average gate dimension based on the average gate fanout, Rent's constant, $\overline{R}$, the effective routing pitch and the number of routing layers.

$$d_g = \frac{f_g \bar{R} p_{eff}}{n_w} \tag{17}$$

STEP 3: The dimension of the chip is the square root of the number of gates times the average gate dimension.

$$D_c = \sqrt{N_g} d_g \tag{18}$$

STEP 4: The average interconnect length, $\overline{R}$, is the average net length in gates, times the average gate dimension.

$$l_{av} = \bar{R} d_g \tag{19}$$

STEP 5: The output resistance for an average gate is the average resistance of a minimum sized transistor, divided by the average size of a transistor.

$$R_{gout} = \frac{R_{tr}}{k} \tag{20}$$

STEP 6: The average gate delay is composed of four terms. The first term is the output loading caused by wiring to other gates. The second term is the delay component caused by input gate capacitance at the receiving gate. The third term models the delay caused by the average signal wire, and the final term is the delay contributed by the input loading of the receiver gates.

$$T_g = f_g R_{gout} l_{av} C_{int} + f_g R_{gout} C_{in} + \frac{R_{int} C_{int} l_{av}^2}{2} + R_{int} l_{av} C_{gin} \tag{21}$$

STEP 7: The maximum clock frequency is set by the total logic path delay, the wiring delay in the clock distribution network and the time-of-flight for crossing the chip. The last term in the denominator includes the speed of light, v_c, and is small, but for large, fast chips, it is now approaching 1% of the total delay.

$$f_c = \frac{1}{T_g f_{ld} + \frac{R_{int} C_{int} D_c^2}{2} + \frac{\sqrt{\varepsilon_r} D_c}{v_c}} \tag{22}$$

STEP 8: Because GaAs DCFL is a constant-current technology, the total power is proportional to the current drawn by the depletion load devices. The depletion devices are sized to deliver about one quarter the current of the pulldown transistors. The total power is the product of the current for each gate, times the supply voltage, times the number of gates. The gate current can be viewed with Ohm's Law as Vcc divided by the pulldown resistance, so the total power is proportional to the power supply voltage squared.

$$P_c = \frac{\frac{k}{4} N_g (V_{cc})^2}{R_{tr}} \tag{23}$$

Parameter	Predicted	Actual
Gate Count	20000	20000
Core Area	68.4 mm^2	68.0 mm^2
Clock period	15.8 ns	7.3 ns
Power Dissipation	10.3 W	11 W

Table 4.5 HGaAs II Model Predictions for Aurora I Chip

Using this model, chip area, clock period and power dissipation were predicted for the Aurora I and II processors. Table 4.5 compares the predictions to the actual parameters for the Aurora I processor. The area and power parameters are quite accurate, but the clock period estimates are significantly higher than the actual values. The entire Aurora I chip was not speed tested. The ALU and register file were the only portions of the Aurora I chip that were tested for speed, so it is likely the true critical path was not exercised.

Table 4.6 lists the model predictions and test results for the Aurora II processor. These numbers are also accurate for the power estimate, but show small discrepancies for the area and again in the clock period estimates. The Bakoglu model can slightly overestimate the clock period because it models only average interconnect length and gate drive to determine speed. Drive transistor sizes on gates in the most critical paths were increased in size to minimize the clock period, causing a slight discrepancy between the actual and predicted values.

4.3 Model Sensitivity

Using the Bakoglu model with parameters appropriate for GaAs microprocessors, one is able to do a first-order evaluation of the various process features for this application. An

Parameter	Predicted	Actual
Gate Count	53000	53000
Core Area	49.7 mm^2	60 mm^2
Clock period	7.6 ns	5.9 ns
Power Dissipation	28 W	24 W

Table 4.6 HGaAs III Model Predictions for Aurora II Chip

parameter	clock cycle sensitivity%	power sensitivity%	area sensitivity%
path length	96	0	0
transistor resistance	96	-100	0
wire pitch	73	0	101
wire cap	73	0	0
driver size	62	-100	0
wire resistance	2	0	0
gate count	11	100	55

Table 4.7 Process Parameter Sensitivity

estimate of the sensitivity of the model to the input parameters is obtained by making small perturbations in each of the input parameters while keeping the others constant. This linearizes the model around a specific design point and allows the parameters with the greatest impact on performance to be identified. Table 4.7 gives the results of the sensitivity study for the HGaAs III process. The table lists the amount of relative improvement for a small change in each of the process parameters. For example, the table shows a 96% sensitivity for critical path length on clock cycle. This means that a 10% reduction in critical path length would reduce clock cycle by 9.6%. A negative sensitivity indicates the parameter moves in the opposite direction from the parameter change. Reducing the transistor ON resistance increases the power dissipation of the chip. Increasing transistor sizes and reducing transistor resistance are obvious methods of improving performance, but do so at the cost of increased power dissipation.

The best ways to improve performance are to reduce the critical path length, reduce the wiring pitch and to reduce the wiring capacitance. Other changes either have small effect, or increase the power dissipation.

4.4 Aurora III Architectural Directions

Process sensitivity analysis in the Aurora II design shows that critical path length is an important design parameter. The final critical path in the Aurora II was 30% longer than

the next most critical path, leading to a very unbalanced design. In addition, the number of logic levels in the two different clock phases was substantially different. Phase 2 had 24 gates in the critical path and phase 1 had only 14 gates, giving a total critical path length of 38 gates.

A target operating frequency of 330 MHz was chosen for the Aurora III chip, based on predicted improvements in the density of the circuit layouts and improvements in the quality of the synthesized logic. A total path length of 28 gates was budgeted for the Aurora III chip, a 35% reduction from Aurora II.

4.5 Aurora III Model Predictions

The Aurora III chip was estimated to require 400K to 600K transistors, with many of these transistors in the on-chip I-cache, and about 75K-gates for logic circuitry. Using 100K gates to represent logic and RAM, the previously developed model can be used to predict the area and performance of the chip, as shown in Table 4.8. With its large number of gates, the Aurora III processor required a reduction in the average transistor size to keep the total power dissipation within manageable limits. The performance model predicts that this decrease reduces performance by 8%. Without the gate size reduction, the chip would have dissipated nearly 70 Watts. The predicted area for the Aurora III processor is acceptable, but the cycle time is significantly higher than the 3 ns target needed to reach a 330MHz clock frequency. The cycle time is dominated by gate delays along the critical path. To improve the performance, it is necessary to increase the circuit density to reduce the average gate loading and speed up these paths.

Parameter	Predicted
Gate Count	100000
Core Area	109.7 mm^2
Clock period	9.51 ns
Power Dissipation	48 W

Table 4.8 HGaAs III Model Predictions for Aurora III Chip

process	core area (mm^2)	cycle time (ns)
HGaAs III	109.7	9.5
HGaAs IV	46.0	6.7
Cray	60.6	6.0
Motorola	31.3	4.6

Table 4.9 Predicted Aurora III Parameters in Different Processes

Table 4.9 shows the predicted areas and cycles times in other processes. Based on work at the University of Michigan and their own research, Vitesse significantly improved the quality of their interconnect for the HGaAs IV process, resulting in much higher circuit densities and higher performance. The primary improvement was a greatly reduced interconnect routing pitch.

The performance model has shown GaAs to have a high sensitivity to interconnect capacitive effects. One reported advantage of GaAs is that the substrate is semi-insulating, giving lower routing capacitance. This is true for analog circuits with few routing layers, but for modern microprocessors with multiple layers of metal and dense routing, the capacitance between layers and between adjacent routes dominants. The metal ground plane that covers the entire design adds capacitance to all routing layers, offsetting any advantage of the insulating substrate.

4.6 Aurora III Model Floorplan

Given the area predictions for the Aurora III, a preliminary floorplan was developed for the HGaAs III process. The chip is composed of five functional units, three of which were assumed to be of similar complexity. The remaining two units together are equal in complexity to one of the other three, which neatly divides the allotted resources into 4 equal parts. Table 4.10 lists the units and their relative sizes.

unit	gates
IEU (Integer Execute Unit)	25%
LSU (Load Store Unit)	25%
IFU (Instruction Fetch Unit)	25%
BIU (Bus Interface Unit)	10%
PFU (Prefetch Unit)	15%

Table 4.10 Area Allocation for Aurora III Processor

Figure 4.3 shows one possible floorplan for the Aurora III processor. Table 4.8 predicts a core dimension of 10.5 mm per side; an additional 1.5 mm is needed for pads and routing, resulting in a final estimated chip size of 12 mm per side.

4.7 Conclusion

Using the high level model, which has been shown to have good correlation with fabricated chips, the critical architectural and process parameters for the Aurora III design were predicted. The model showed that if the processor were fabricated in the originally intended process, it would not meet a 330MHz performance target. Routing requirements

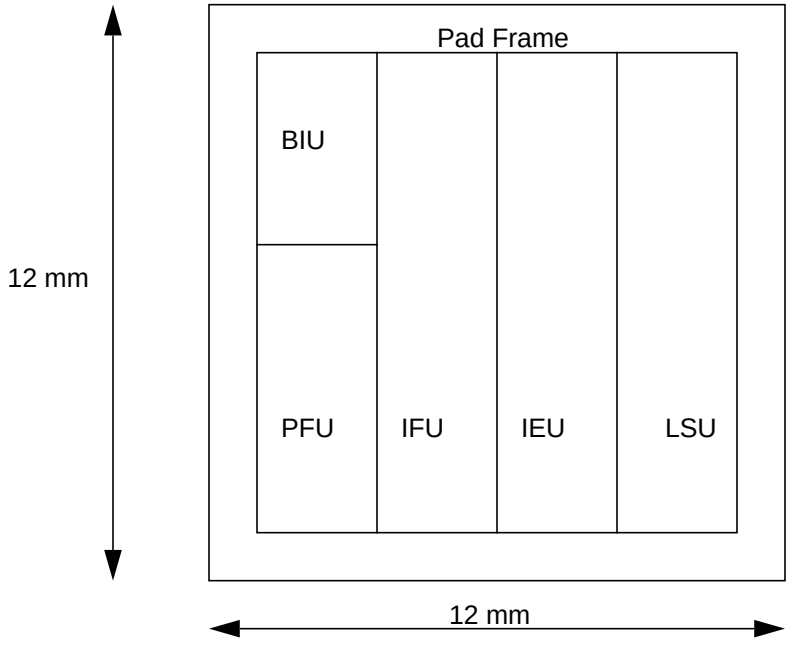

Figure 4.3 Target Aurora III Floorplan

were defined, and a new process was proposed by the target fabrication vendor to meet the interconnect needs of the chip.

CHAPTER 5

Impact of GaAs Technology on Architecture

The previous chapter analyzed the sensitivity of gallium arsenide technology to various process parameters, and provided rough performance targets for the Aurora III microprocessor. These performance targets could not be reached without significant improvements in the chip-level components used. This chapter discusses several circuits techniques developed for high-speed GaAs processors.

5.1 Path Length Reduction

The sensitivity analysis verifies that clock rate can best benefit by reducing the number of gates on the critical paths. Our group at the University of Michigan devoted a substantial effort to developing architectural components that would reduce critical path length. A second focus was on improving the density of the circuit layouts, because higher density results in lower parasitic loading.

5.2 Interconnect Parasitics

In the past, the technology driver for process development was the dynamic RAM. Microprocessors have recently emerged as a process driver in their own right, with a significantly different set of design criteria. The primary concern for RAM design is that the area of the RAM bitcells be minimized. While area is an important criterion for microprocessors, the high power and fast clock frequencies place additional demands on the process, particularly on the metallization system.

Microprocessors demand a low-skew clock distribution network. To achieve low skew the interconnect impedance must be minimized. Because of the thickness of the low impedance wire, it is difficult to provide both dense routing and low wire resistance. Most processes compromise, providing several high-density layers with high routing resistance and

a thick metal layer with lower routing density for clock and power distribution.

5.3 Functional Decomposition

The system performance models show a decrease in chip operating frequency as the gate count is increased. Two factors contribute to this increase: additional clock distribution overhead, and an increase in average gate delay caused by a longer average net length. This second factor, which is multiplied by the number of gates on the critical path, can be significant. Partitioning the design into smaller components reduces the average gate delay for the partitioned design by reducing the average interconnect length. The model assumes an amorphous design style, consisting of only one large block that implements the entire design. Modern architectures, however, are designed in a more modular and hierarchical style. In a modular design, changes in the complexity of one module have little impact on the performance of other modules.

The performance of a modular design is set by two parameters, the maximum delay of any of the modules, and the maximum delay of any intermodule signals. In a pipelined processor, it is usually possible to pipeline the communication between modules so that these communication signals are not on the critical path. Although there are relatively few global signals, their loads are so much larger and their lengths so much longer than the average signal that special high-power drivers are often necessary to ensure correct timing. To minimize power dissipation, the number of signals requiring special treatment should be minimized.

Performance of the modular design can be estimated by finding the slowest of the constituent modules. Because each module is smaller, the average net length and power are reduced, and the clock frequency is increased. Increasing the modularity of the design has costs as well as benefits. Additional clock cycles are needed to transmit signals between modules. Assuming that intermodule delays are not a factor, the system performance model shows a 13% improvement in clock frequency when the design is partitioned into four equal-sized components.

5.4 Circuit Design Techniques for Reduced Path Length

The circuit restrictions imposed by DCFL technology greatly restrict innovative circuit solutions to design problems. Other technologies have more gate types and logic styles that can be applied to improve performance. In DCFL, any circuit may be used as long as it is a NOR gate of four or fewer inputs. This has forced us to look to other approaches for improving clock speed.

High-current gates use Feedback FET Logic (FFL) to limit the output current and prevent overdriving the receiving gate inputs [Fulkerson91] An FFL inverter schematic is shown in Figure 5.1. The buffer operates like a NMOS superbuffer. When the input is high, the pulldown FET is on and the output is turned off. When the input is low, the pulldown FET is off, and the pullup FET is on. As the gate output rises to a logic ONE, the feedback FET turns on, draining current from the pullup inverter.

As discussed in Chapter 3, the logic synthesis library for the Aurora I processor did not have high-current buffers. Buffers were added manually to drive heavily loaded lines. These gates accounted for 30% of the standard cell gates used in the Aurora I chip. Figure 5.2 shows the high drive gates as shaded symbols. On the left is the Aurora I example with low-drive gates in the synthesized module, and the high-drive gates added to increase the drive capability.

To reduce the gate count and logic depth, these buffers were replaced with high-current

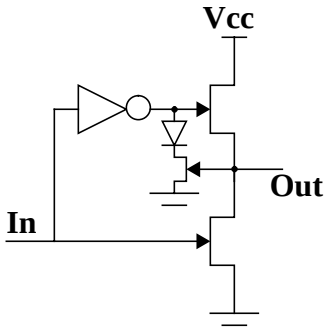

Figure 5.1 Feedback FET Logic Buffer Schematic

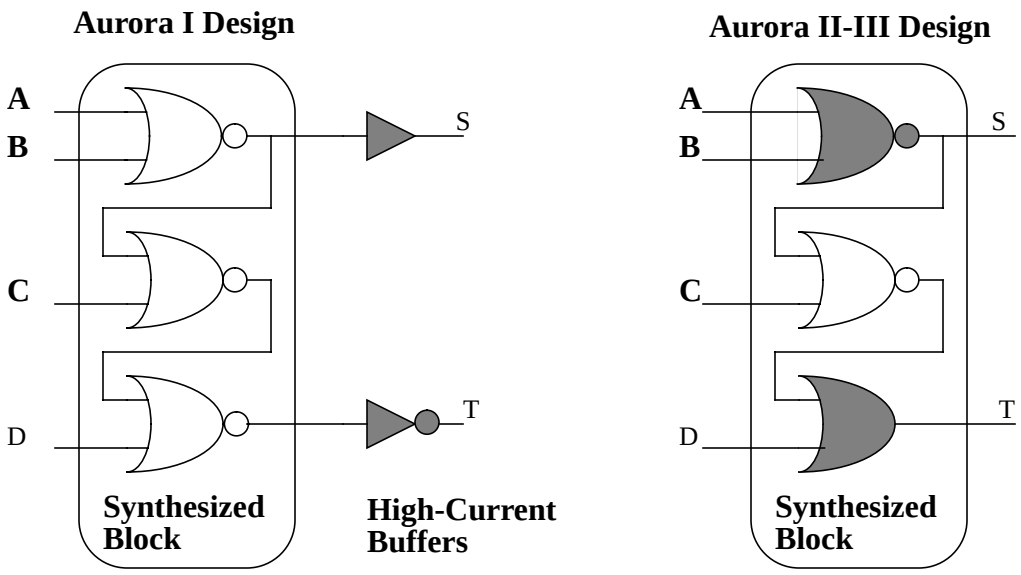

Figure 5.2 High Drive Logic Synthesis Outputs

logic gates on later chips. The logic first was synthesized using a restricted library that did not have high-current buffers, then the gates driving all exported signals from the synthesized block were replaced with the high-current equivalent. Gates driving only local signals were not modified. This optimization reduced both the total cell count and the number of logic levels for all control signals and was a major factor in the increased layout density.

Commonly occurring logic groups were identified in the Aurora I design for optimization. The most frequent logic combination was a 2-to-1 multiplexor followed by a latch and a high current buffer. Using the Earle latch technique [Kogge81] these three functions were merged into a single module, reducing the number of logic levels for this function from 9 to 6.

In the Aurora II chip, the pipeline datapath vertical pitch was too small to accommodate the many data busses that passed between modules. A major contributor to the congestion was the 4-to-1 multiplexor used in the bypass network and in other places. The four bus

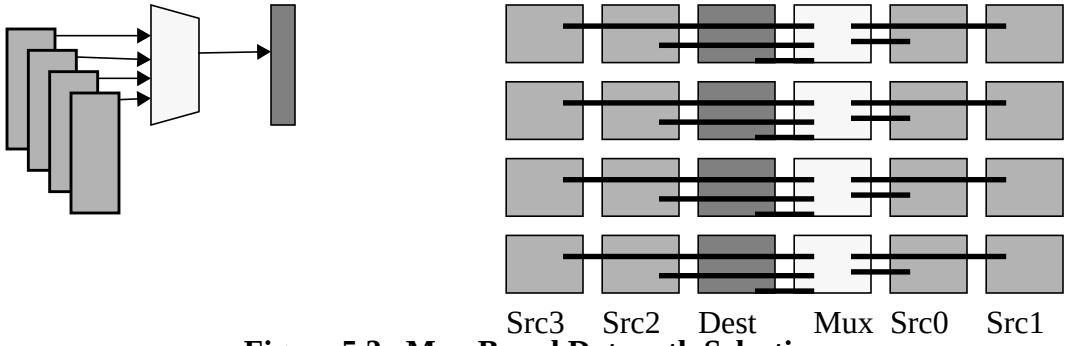

Figure 5.3 Mux Based Datapath Selection

inputs and one bus output need a minimum of three routing tracks. When two muxes were adjacent in the datapath, more than half of the available routing tracks were consumed. Figure 5.3 shows the logical and layout views of part of a four-bit datapath. In this example, a multiplexor selects the output of four Phi 1 latches as input to a Phi 2 latch. A total of five busses are needed, four for the inputs to the mux and one for the input to the Phi 2 latch. These five busses are shown in the layout view. The module shading indicates the type of module; the busses are shown as dark wires. Note there is no way to order the cells so that fewer than three routing tracks per cell are required.

A tristate buffer was developed to solve the congestion problem. Instead of using a mux to select one of N possibilities, a tristate driver was added to each of the possible sources. The output of all tristates are tied together to form the input to the next latch. Figure 5.4 shows the logical and layout views of the tristate-based solution. In this case, only one bus

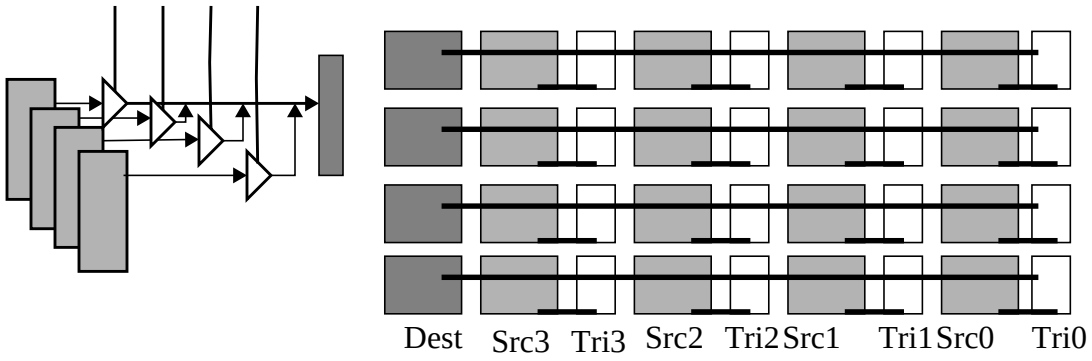

Figure 5.4 Tristate Based Datapath Selection

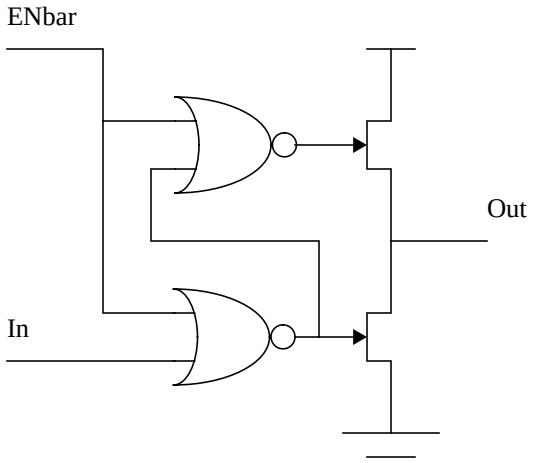

Figure 5.5 Tristate Buffer Circuit Diagram

and local interconnect are needed, requiring at most two routing tracks rather than three. Figure 5.5 shows the tristate buffer circuit design. When the signal ENbar is a logic ZERO, the buffer works as a normal high drive buffer cell. With the ENbar signal a logic ONE, the outputs of both NOR gate drivers are brought low, turning off both drive transistors.

5.4.1 Ling Adder

Fast addition is a vital component in any high performance microprocessor architecture. Early microarchitectural studies evaluated several different 32-bit adder architectures, including carry-skip [Turrini89], carry select [Bewick88] and 4-bit lookahead schemes. The adder design chosen is based on the work of Quach and Flynn [Quach90], and uses a Ling adder carry chain combined with a conditional-sum final stage.

The Ling adder was designed to take advantage of the wire-OR capability of ECL logic [Ling81]. In ECL, a 32-bit sum can be generated in 3 gate levels of two-level series-gated logic. As GaAs lacks both wire-OR and complex gate capability, it was not clear that this architecture would match well with the circuits realizable in GaAs DCFL. A high-level macro-model (based on that developed by Johnson at MIPS [Stritter90]) was written in C for both the Ling adder and the 4-bit carry lookahead adder.

The results of the Ling adder simulations showed that while the adder took more gate levels in GaAs than in ECL, the Ling scheme did have a significant performance advantage

over a more conventional 4-bit group lookahead. Both the estimated delay and the number of logic levels were reduced by 30%. The Ling adder critical path was 10 gates long, compared to 14 in the 4-bit lookahead adder. Fan-in limits required the use of 3-bit carry lookahead groups instead of the more typical 4-bit groups. The conventional carry lookahead scheme uses the following relations:

$$P_i = A_i \oplus B_i \tag{24}$$

$$G_i = A_i B_i \tag{25}$$

$$S_i = P_i \oplus C_{i-1} \tag{26}$$

$$C_i = G_i + P_i C_{i-1} \tag{27}$$

$$GG_i = G_i + P_i G_{i-1} \tag{28}$$

Exclusive OR is represented by the symbol $\oplus$, inclusive OR by the plus sign +, and logical AND by adjacency. A slight modification of the previous terms results if the inclusive OR is used in place of the exclusive OR.

$$P_i = A_i \oplus B_i \tag{29}$$

$$T_i = A_i + B_i \tag{30}$$

$$G_i = A_i B_i \tag{31}$$

$$S_i = P_i \oplus C_{i-1} \tag{32}$$

$$C_i = G_i + T_i C_{i-1} \tag{33}$$

$$GG_i = G_i + T_i G_{i-1} \tag{34}$$

The second formulation generates the carry term faster at a cost of a slight addition in logic. In nearly all logic families, the inclusive OR is faster to generate than the exclusive OR. The true benefit of the Ling scheme can be seen when both of these approaches are extended to multiple-bit groups. In the 3-bit carry lookahead adder method, the intermediate group generate (GG) terms are calculated in parallel by cascading Equation (28) N bits wide. First, the GG terms for the normal 3-bit CLA will be developed as follows:

$$GG_{i+2} = G_{i+2} + P_{i+2}G_{i+1} + P_{i+2}P_{i+1}G_i \tag{35}$$

$$GG_{i+2} = (A_{i+2}B_{i+2}) + (A_{i+2} \oplus B_{i+2})\,(A_{i+1}B_{i+1}) + (A_{i+2} \oplus B_{i+2})\,(A_{i+1} \oplus B_{i+1})\,(A_iB_i) \tag{36}$$

$$GG_{i+2} = (A_{i+2}B_{i+2}) + (A_{i+2}A_{i+1}B_{i+1}) + (B_{i+2}A_{i+1}B_{i+1}) + (A_{i+2}A_{i+1}A_iB_i) + (A_{i+2}B_{i+1}A_iB_i) + (B_{i+2}A_{i+1}A_iB_i) + (B_{i+2}B_{i+1}A_iB_i) \tag{37}$$

In the Ling scheme, the GG term is written in terms of T rather than in terms of P:

$$GG_{i+2} = G_{i+2} + T_{i+2}G_{i+1} + T_{i+2}T_{i+1}G_i \tag{38}$$

Observing from Equation (30) and Equation (31) that $G_i = G_iT_i$ the GG term can be re-written:

$$GG_{i+2} = T_{i+2}G_{i+2} + T_{i+2}G_{i+1} + T_{i+2}T_{i+1}G_i \tag{39}$$

$$GG_{i+2} = T_{i+2}(G_{i+2} + G_{i+1} + T_{i+1}G_i) \tag{40}$$

$$GG_{i+2} = T_{i+2}(A_{i+2}B_{i+2} + A_{i+1}B_{i+1} + A_{i+1}A_iB_i + B_{i+1}A_iB_i) \tag{41}$$

$$H_{i+2} = A_{i+2}B_{i+2} + A_{i+1}B_{i+1} + A_{i+1}A_iB_i + B_{i+1}A_iB_i \tag{42}$$

$$GG_{i+2} = T_{i+2}(H_{i+2}) \tag{43}$$

As seen in equation (37), for the 3-bit CLA adder the GG term can be generated from the A and B inputs using 7 terms with 24 total inputs. The Ling adder propagates the carry in terms of H_{i+2}, a signal generated from 4 terms of 10 total inputs. The H_{i+2} terms can be generated directly from the A and B inputs with two levels of NOR gates.

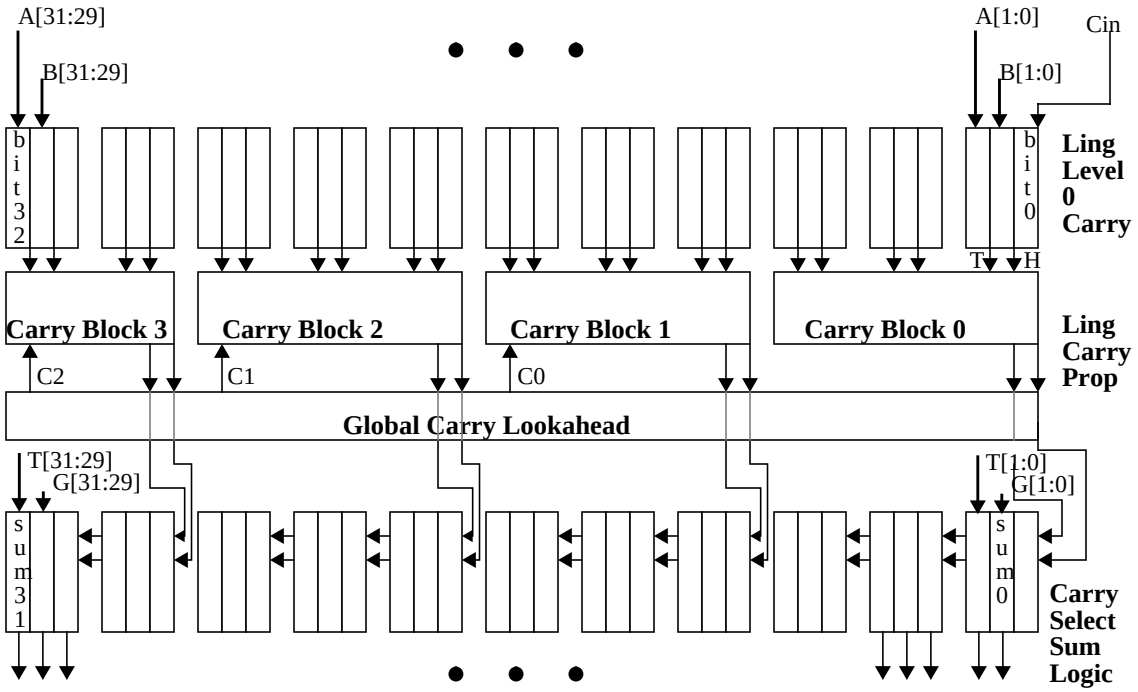

Figure 5.6 Ling Adder Block Diagram

Figure 5.6 shows a block diagram of the 32-bit GaAs Ling adder. The adder uses 3 bit Ling carry groups to generate the first level of carry signals in two gate delays in the level 0 carry blocks at the top of the diagram. These first level carry signals then are combined in the next level to generate intermediate carries for groups of 9 bits (6 bits for the last stage). These intermediate carries are combined to form the top level of the carry tree.

The top level of the carry tree and the intermediate block carries are used in a two-level carry selection scheme to produce the final sum bits in 13 gate delays.The GaAs Ling adder has 4 fewer logic levels than the 3-bit CLA. Two of these levels are saved by using the directly generated H terms for the carry propagation. The other two levels are saved by using a 2 level carry select scheme to generate the final Sum values.

5.4.2 Pipelined Ling Adder

ALU operations in the Aurora II processor produced their results in one clock phase, requiring the entire addition be calculated in 1/2 clock cycle. To achieve the target clock

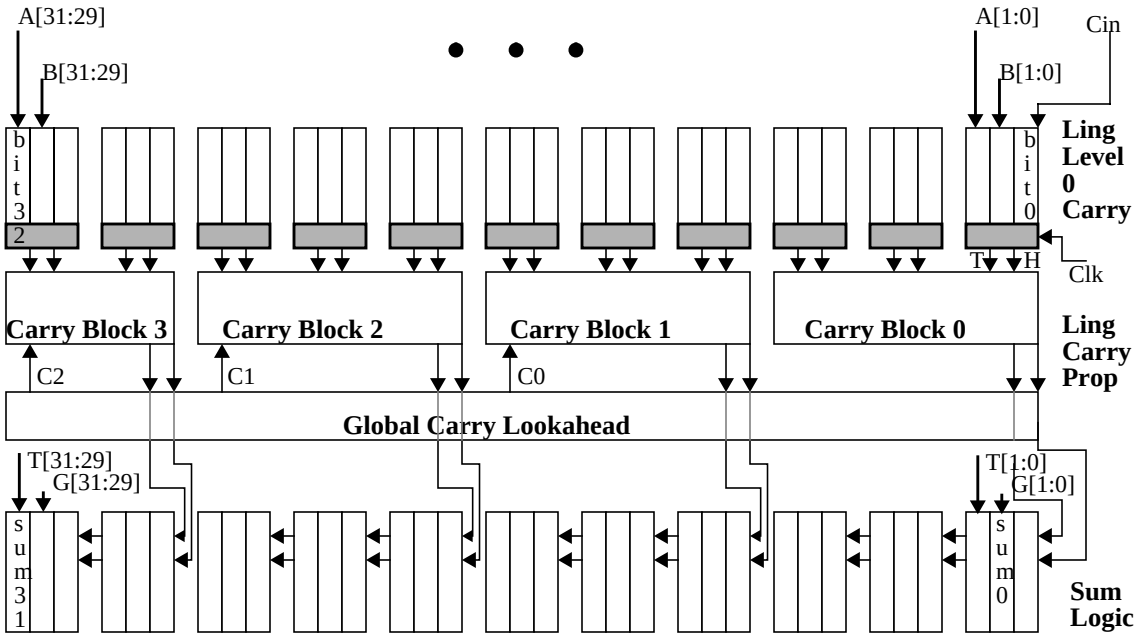

Figure 5.7 Pipelined Ling Adder

frequency for the Aurora III, it was necessary to pipeline the ALU addition, splitting the add across two clock phases. The same concept used to create the Mux-Latch-Buffer cell can be extended to include other logic functions in place of the multiplexor. The logical place to pipeline the add was after the first stage of carry generation, latching the T, H and first-stage carry signals. The positions of the added latches are indicated by the shaded boxes in Figure 5.7.

5.5 Summary

This chapter described the importance of reducing critical path length and interconnect parasitics. A modular architecture was proposed, and shown to have 13% higher performance than a unified architecture, due to reduced bus parasitics. A method for synthesizing high-current control signal drivers was presented. Circuit design techniques for high speed adders and bussing structures were described. These techniques combine to greatly improve the quality of circuits possible in DCFL.

CHAPTER 6

Aurora III Microprocessor System Architecture and Design

6.1 System Overview

The Aurora III microprocessor is the culmination of four years of research on the effective utilization of Gallium Arsenide technology for the construction of pipelined microprocessors. A block diagram of the Aurora III system is shown in Figure 6.1. The system is composed of 4 custom GaAs chips, three logic chips and a 32K bit SRAM for building a 64k byte data cache. The three logic chips are the Floating Point Unit (FPU), Integer Processing Unit (IPU) and the Memory Management Unit (MMU). A distinguishing feature of the Aurora III system is the extensive use of pipelining, streaming data fetches and memory queues to decouple the various elements of the design. The challenge in building a high-

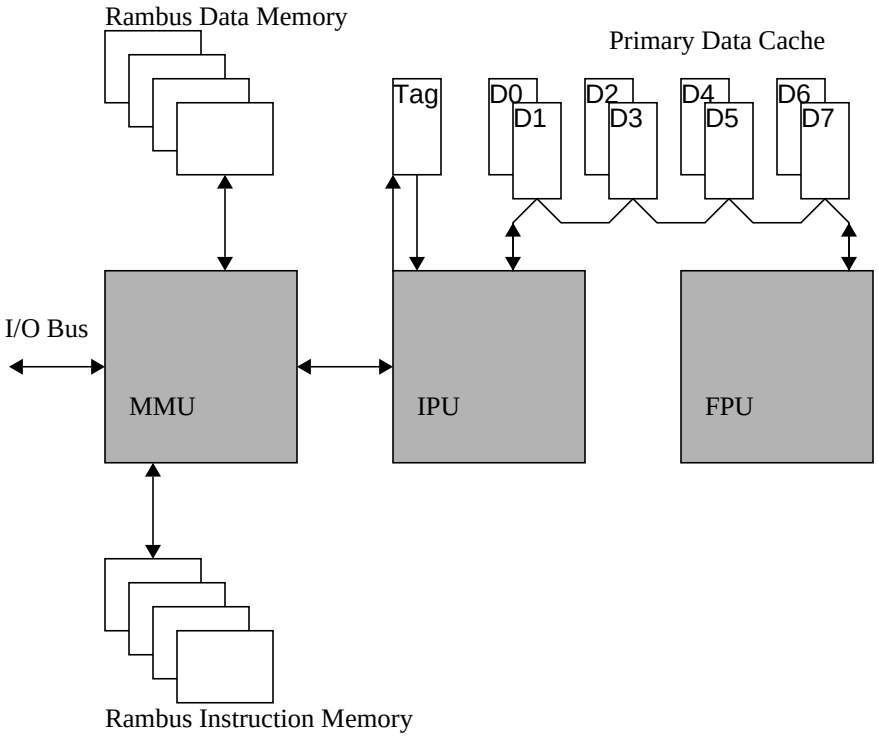

Figure 6.1 Aurora III System

throughput computer lies not so much in performing the computations, but in supplying the data needed for the computations and retiring the resulting data.

Many of the architectural features of today's microprocessor first appeared decades ago in large-scale computers. Many of the new features in current microprocessors can be traced to these old mainframes and supercomputers. Figure 6.2 shows contemporary microprocessors which embody the indicated Aurora III architectural features. In some cases these processors were the inspiration for inclusion of the features in the Aurora III; in many other cases the design activities were concurrent, and were inspired by literature studies and similar design constraints. A timeline is shown on the right of the figure. The vertical position of the different processors indicates the approximate date when the design became public knowledge. The Aurora III processor inherits many of its attributes from three primary sources. The instruction set architecture is derived from the MIPS R3000, and the

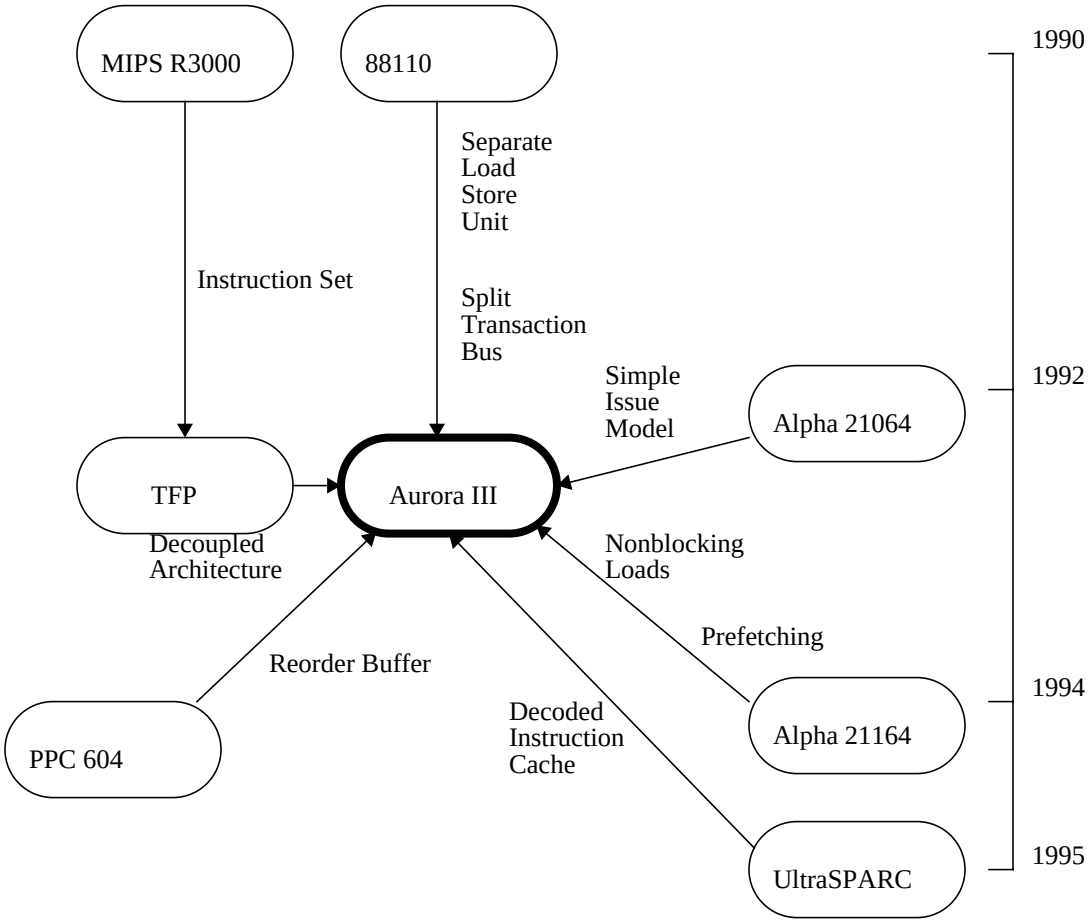

Figure 6.2 Architectural Heritage of the Aurora III Processor

pipeline model is derived from the Motorola 88110. A simple issue model is adopted to maintain a high clock rate, as in the DEC Alpha 21064 processor.

Although processors have recently appeared with features similar to Aurora III, the long time required for processor design means that those processors were developed concurrently. For example, the chip partitioning and design goals for the Aurora III and the SGI TFP processor are nearly identical, but were developed independently from a similar set of design criteria. Aurora III adopted many of the advanced memory system ideas not from existing machines, but through the academic literature.

Trace driven simulation was used to evaluate various processor architectural features. Architectural performance was evaluated using the integer and floating point Spec92 benchmarks. Time constraints imposed by other phases of the design process limited the length of the benchmarks that could be run. The integer benchmarks used the “small test” input file, and the floating point benchmarks were limited to their first 90 million cycles. In all, about 176 million instructions were run from the integer suite, and about 810 million from the floating point suite. All benchmarks were compiled using GCC with no additional code rescheduling.

The XSIM trace-driven simulation program from Stanford University was used to read memory trace files [Smith91]. A cycle-accurate pipeline model was incorporated into the XSIM model to calculate cache hit rates and cycle counts for the benchmarks. The simulator was written in C++ and contains about 10,000 lines of code. The sizes and latencies of all internal memory structures are easily varied to study the effects of such changes on system performance.

The data cache and floating point unit are connected to a common set of pipelined data buses, as shown in Figure 6.1. There is a three-cycle access delay when the integer processing unit requests data from either of these components. However, because both the data cache and the FPU are pipelined, a new access can begin each cycle.

The busses from the CPU to the cache and FPU are unidirectional with respect to the IPU and cache. An IPU output bus transmits data from the IPU to the cache and FPU, and

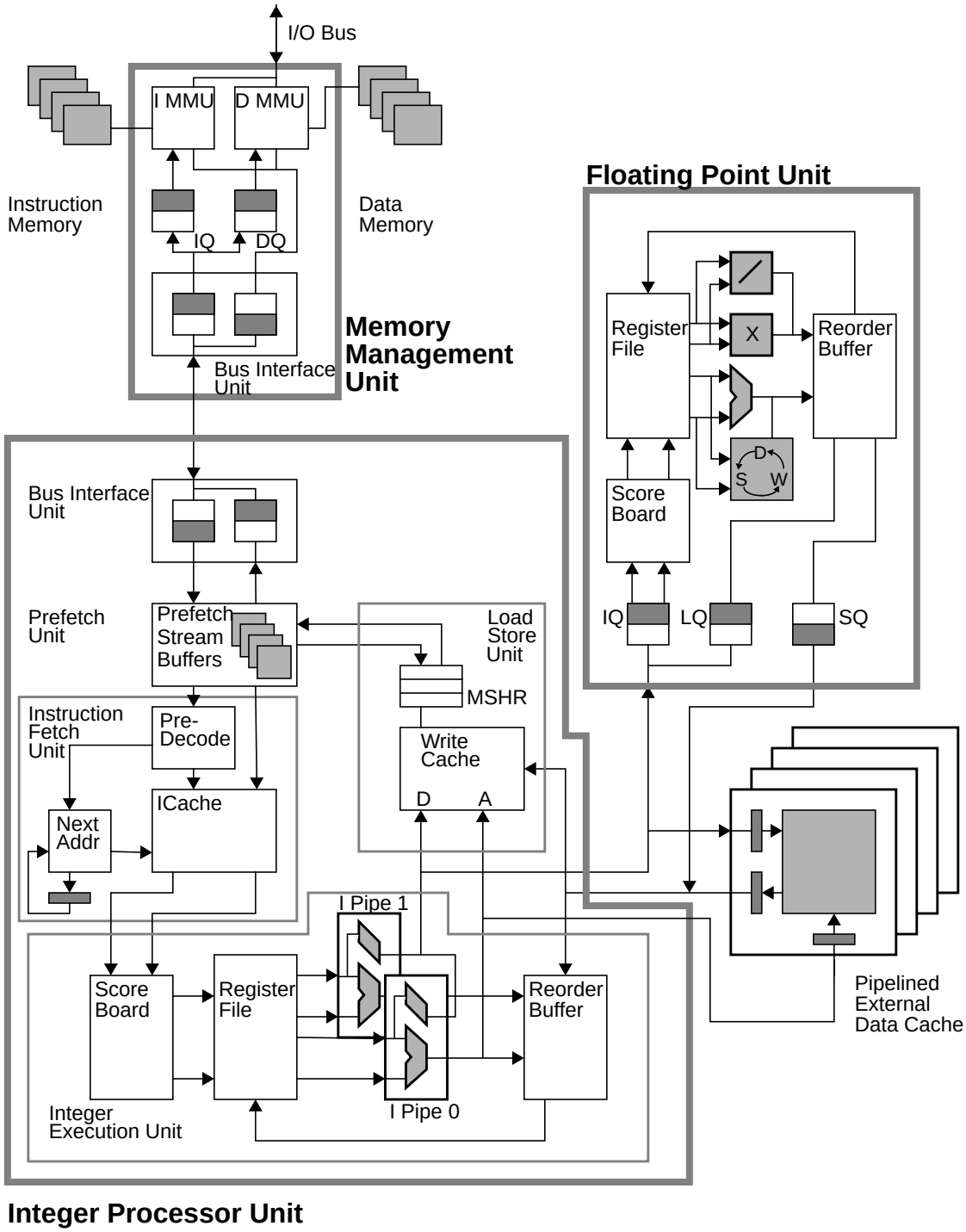

Figure 6.3 Processor Block Diagram

is also used to transmit data from the FPU to the cache. An IPU data input bus receives data from both the cache and FPU, and allows the cache to send data directly to the FPU. This bus arrangement requires the FPU to support bidirectional data transfers on both data busses.

The Aurora III memory system is designed to provide a large bandwidth from main memory through the MMU, and to the primary caches. Each component can support a peak transfer rate of 64 bits each processor clock, giving a maximum memory bandwidth of 2.4 G-bytes per second. Sustained transfer rates of over 1 G-byte per second have been achieved in the system simulation model.

6.2 Processor Organization

The IPU consists of five functional modules that operate semi-autonomously to fetch, decode, execute and retire instructions. The IPU is similar to the IBM-Motorola PowerPC 603 and 604 processors in that it includes a Bus Interface Unit (BIU), an Integer Execution Unit (IEU), an Instruction Fetch Unit (IFU), and a Load Store Unit (LSU). In addition to these, the IPU has a dedicated Prefetch Unit (PFU) for data and instructions. Figure 6.4 shows the high level organization of the components.

For the system level performance of our GaAs chipset to be competitive with CMOS processors, the GaAs system must overcome with increased clock frequency the CMOS advantage of much higher integration density, which provides increased parallelism and larger caches to make up for a lower clock speed. The Aurora III research explores microarchitectures that allow parallel instruction issue with the goal of maintaining a fast processor cycle time.

The high-level performance models discussed in Chapter 4 pointed to two factors as being important in achieving high system clock frequencies. The first critical factor was minimization of the number of logic levels needed to implement a function. The second was minimization of the area required to implement a given function. Among the main contributors to the die area in our previous chips were pipeline latches included to increase the op-

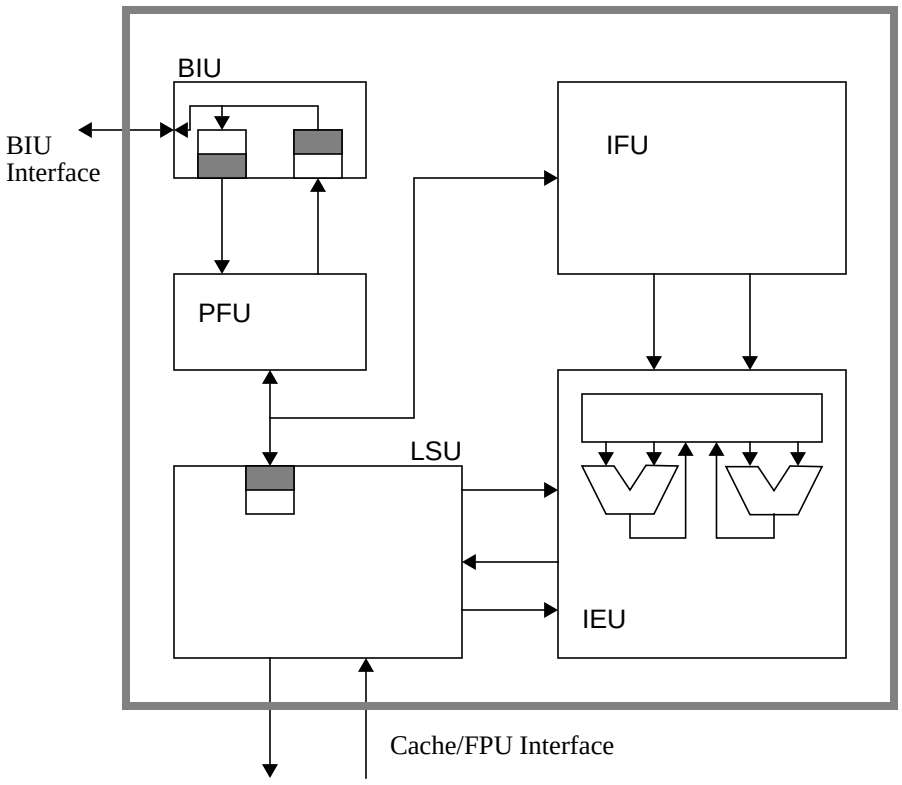

Figure 6.4 Integer Processing Unit Block Diagram

erating frequency by dividing long paths into shorter units. Every pipeline stage after the ALU, such as load delay slots, requires a forwarding path back to the ALU inputs. These latches and forwarding paths consume a large amount of area, increasing the parasitic loading of the nets. The state latches and forwarding network for the Aurora II design accounts for nearly 50% of the execution pipeline area.

One way to minimize the area penalty for state latches is to have a short execution pipeline. This implies one of two things, either the cycle time is slowed to eliminate the load delay slot, as is done in Sun's SuperSPARC chip, or integer computation instructions and memory instruction must have different length pipelines. The second approach was adopted for Aurora III. Figure 6.5 shows the pipeline stages for integer and memory instructions. Integer computations require four cycles. Memory instructions require an additional two or three cycles to produce their results. Instructions are read from an on-chip instruction cache in the IC stage. Register operands are read from the register file in the RF stage. Computa-

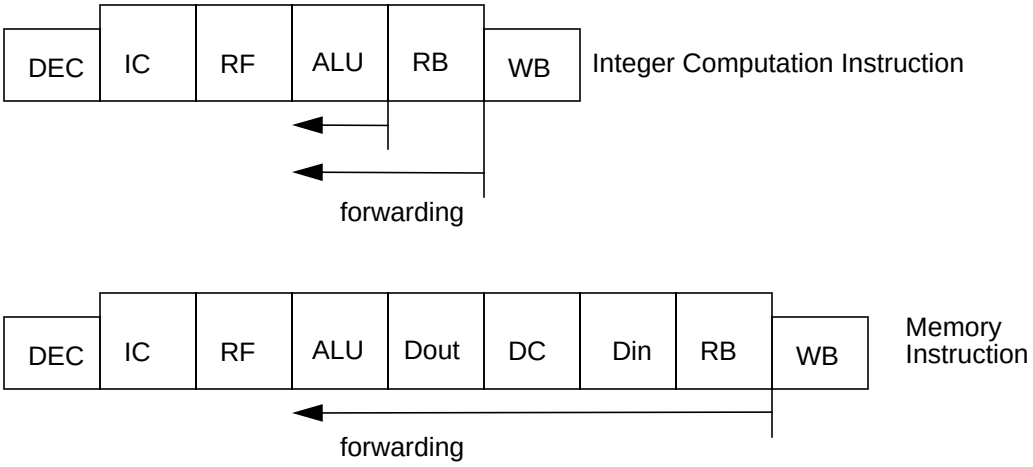

Figure 6.5 IPU Pipeline Forwarding Paths

tion is performed in the ALU stage, and results are written into the reorder buffer in the RB stage. After the RB stage, because of forwarding, the results appear to the rest of the machine as to have been written to the register file, although the actual write back is delayed until the WB stage. Forwarding paths allow the ALU output and the reorder buffer contents to be used as inputs of the next instruction, causing no pipeline bubbles. Instructions coming from the BIU interface must pass through one stage of predecoding, but instructions that hit in the on-chip instruction cache have previously been decoded, and skip this stage.

Memory instructions require an additional 3 cycles. The virtual address is generated in the ALU stage. The address and any store data are transmitted to the data cache in the Dout stage. The cache is accessed in the DC stage, data are returned from the data cache to the IPU in the Din stage, and data are written to the reorder buffer in the RB stage, at which time it is available for subsequent instructions. The Aurora III IPU pipeline structure is shown in Figure 6.6.

The IPU is designed to support a range of process technologies having different speeds and integration levels. The modular design of the IPU allows easy customization of the sizes of a variety of internal memories. The asynchronous BIU interface allows external communication and internal computation to proceed concurrently. The following sections

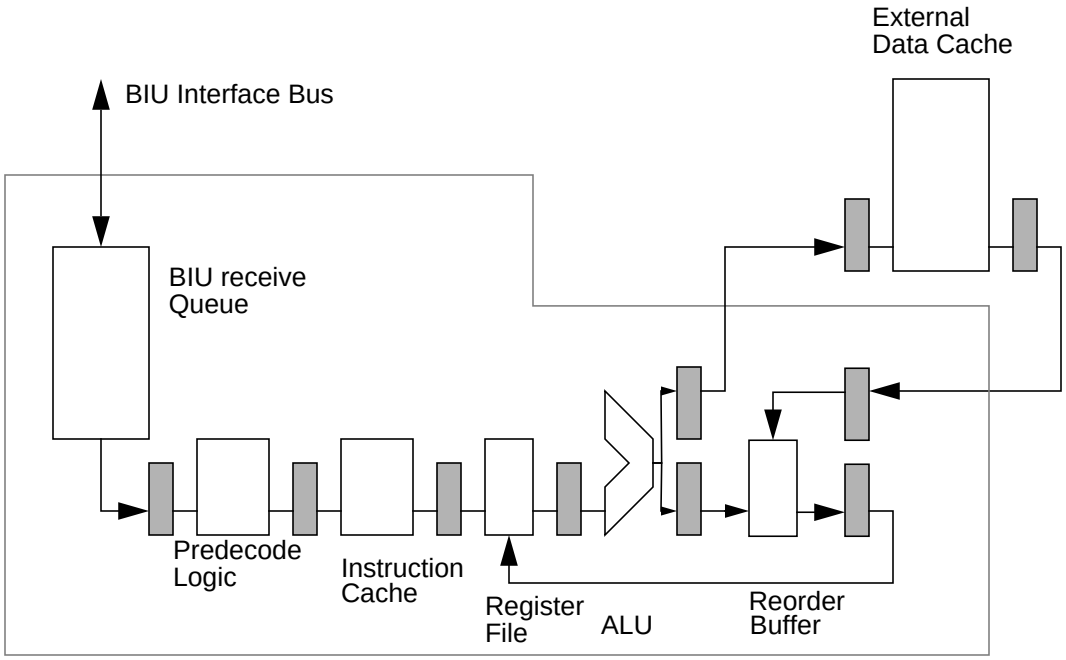

Figure 6.6 IPU Pipeline Stages

present detailed descriptions of the IPU function units.

6.3 Instruction Fetch Unit

The IFU fetches instructions and maintains the state of the on-chip Instruction Cache. I-cache misses are detected in the IFU, and the missing instructions are requested from the MMU via the BIU. The front of the IEU pipeline stalls until the needed instructions arrive, but the LSU continues to process pending data cache misses, and the reorder buffer continues to retire completed instructions.

An on-chip instruction cache was required to support a two instruction-per-cycle issue rate. Too few I/O pins were available to fetch 64 instruction bits each cycle from off-chip. There are few restrictions on which instructions can issue together. In addition to true instruction dependencies, in which an instruction uses the result of the immediately preceding instruction, the primary issue constraint is that only a single memory access instruction can be executed in a given cycle.

Decoded Instruction Cache

To speed the instruction issue logic, instructions are predecoded before being inserted into the instruction cache. Figure 6.7 shows the arrangement of a decoded instruction. All instructions are grouped into pairs, with the EVEN instruction occupying the lower of two consecutive addresses, and the ODD occupying the next sequential address. The DI bit indicates whether an instruction pair dependency prohibits dual issue. The CONT field indicates whether the instruction pair includes a control flow instruction, such as a jump or branch. The MIPS ISA prohibits a branch instruction in a branch delay slot, so there will be a maximum of one control flow instruction in each pair. To simplify the design, all branch instructions are executed by the EVEN pipeline.

The static pairing of instructions is a more restrictive issue model than that employed by many other processors. If a pair of instructions can not dual issue, they are issued over two clock cycles, and then the following pair is examined. In some processors, such as the Supersparc, the instruction following the current pair of instructions is checked to see if it can dual issue with the current pairs ODD instruction. This creates many critical speed paths in the design, and was rejected as a design option for Aurora III.

Next Address Caching

If the instruction pair contains a control flow instruction, the NEXT field contains the cache index of the target instruction. This next address caching (or branch folding [Ditzel87]) reduces the critical path for a dual issue machine by eliminating the branch pipeline bubble. The target of the branch can be fetched on the next cycle, without needing to compute a target address from the address of the branch instruction.

Next address caching is an extension of an old microcode idea. In some machines, each microword contained the address of the next microword [Husson70]. The next address field was fed directly into the micro-instruction address port. Concurrent with the Aurora III

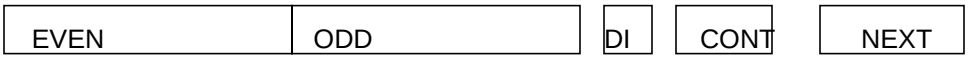

Figure 6.7 Decoded Instruction Cache Format

project, SGI and Sun developed a similar method for next address caching on the TFP and UltraSPARC processors [Hsu94, Agrawal94].

Branch Prediction

One bit in the predecoded I-cache CONT field contains a static branch prediction. The prediction algorithm is user configurable, based on two bits in a configuration register. The possible algorithms are predict not-taken, predict forwards-taken, and predict backwards-taken.

The branch prediction could be easily extended to a dynamic algorithm with little increase in complexity for the instruction decode logic. Updating the prediction bits when the state of the prediction must change requires writing to the I-cache. With a two-bit Smith predictor these updates should be rare, and pending updates could be queued until spare cycles are available for modifying the I-cache [Smith81].

The program counter datapath calculates the predicted and non-predicted branch targets. The predicted path flows down a program counter pipeline to place the instruction address in the reorder buffer to allow exception recovery. The non-predicted address is also pipelined, allowing single-cycle branch misprediction recovery. Immediately upon the detection of a mispredicted branch, the address at the end of the non-predicted pipeline initiates the fetch of the correct branch target address; no additional branch address calculations are needed.

6.4 Execution Unit

Dual, short, full-function pipelines

Some recent machines restrict the types of instructions that each execution pipeline can perform. This method was evaluated and rejected because the reduction in complexity in the execute units was offset by an increase in complexity in the instruction issue logic. The issue logic was known to be a primary performance bottleneck, so complexity in this stage was avoided even at the expense of a slight logic increase in the execute units. One benefit of this decision was that only a single execute unit design was needed. Had two different

execute units been used, design effort and optimization time would have been divided between them.

Scoreboard

The key elements of the Instruction Execution Unit (IEU) are the integer register file, two execution pipelines and a six-entry reorder buffer. A register file scoreboard [Thornton70] is used to detect instruction dependencies and stall execution until the needed operands are ready. As each instruction is issued, the destination register for that instruction is marked as busy. The pipeline will stall if a subsequent instruction tries to read a busy register before the result for that register has been computed.

ALU

The ALU is composed of a logic unit, a high speed barrel shifter and a 32-bit adder. The pipelined Ling adder is described in Chapter 5. Pipelining the addition across two clock phases equalizes the critical path lengths. The first phase logic in the adder is also used to compute the logical operations, greatly reducing the cost of the logic unit. The shifter is made from a cascade of 3-input multiplexors. Five levels of multiplexing are used, each level shifting left or right by increasing powers of two, or passing the bits through unshifted. A final multiplexor selects which of the units is latched as the result. This multiplexor could have been merged with the final carry-select logic in the adder to reduce the delay, but the ALU was not the final critical path, so this was left as an enhancement to be added later if necessary.

Speculative Execution and the Position of the Control Point

Predicted branches are resolved after the ALU stage, three cycles after instruction fetch. A mispredicted branch will squash all instructions currently in the pipeline and initiate an instruction fetch at the correct target address. Squashing the instruction in the pipeline allows multiple speculative branches to be present in the pipeline simultaneously. Because branches can be processed each cycle, there may be as many as three speculative branches being processed at once. Instructions are issued in order; only the state in the machine before the ALU stage is speculative. No speculative instructions are sent to the LSU or com-

mitted to the reorder buffer; this greatly simplifies the control logic.

Bypass Network

The execution pipeline provides a general bypass network capable of forwarding internal results from either pipeline or the reorder buffer to any of the source register inputs. The bypass network in previous chips were built from multiplexor logic, increasing wiring congestion and lengthening the critical paths. The bypass network in the Aurora III uses the tristate buffer described in Chapter 4 to increase the logic density and reduce the critical path length. The new design was important because Aurora III's increased parallel units have more possible bypass paths.

Reorder Buffer

A reorder buffer allows instructions to complete out of order while still maintaining precise exceptions [Smith85b]. The reorder buffer provides additional architectural benefits. To support dual instruction issue, a total of 4 reads and 3 writes are needed each cycle. Providing this number of ports in a single register file increases the register file critical path and creates a routing bottleneck. To avoid this problem, the design includes a 2-read 1-write register file in each execution pipeline. The contents of these two register files are kept identical by simultaneously writing results to both. The reorder buffer has 3 write ports, one for each pipeline and one for load data returned from the Load/Store Unit.

Precise Exceptions

When the execution of an instruction is completed in a pipeline, the result is written to the reorder buffer. The reorder buffer reserves space for these results in the order that instructions were issued; on completion of the instruction, the data is written to this reserved location. After the data are written to the reorder buffer and all possible exceptions have been resolved, they are copied in program order to the register files. Both copies of the register file receive the same write data from the reorder buffer.

The reorder buffer supports full bypassing of intermediate results to both pipelines. To provide forwarding, the reorder buffer needs 5 read ports and 3 write ports, as would a register file designed to support the same level of concurrency. However, the reorder buffer

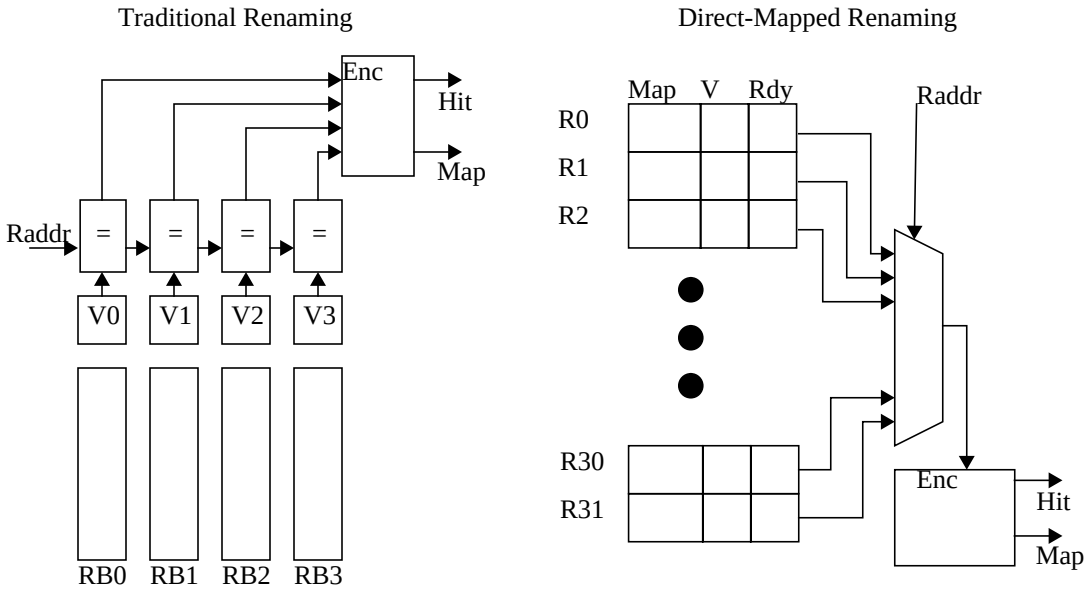

Figure 6.8 Traditional vs. Direct-Mapped Register Renaming

contains only 6 registers, as opposed to the 32 required for the full register file. A secondary benefit of the reorder buffer is that it allows multiple live copies of a register to exist in the reorder buffer at once. This removes the Write-After-Write conflict from restricting instruction issue.

Register Renaming

Register renaming using reorder buffers is often implemented using an associative look-up of the register tags in the reorder buffer to detect renamed registers. If a register has a valid entry in the reorder buffer, the value in the register file is out of date and should not be used. This look-up is complicated because there may be more than one mapping for a register. In this case the latest value must be used.

Previously this logic has been designed using a priority-encoded Content Addressable Memory (CAM) structure for the reorder buffer register tags. This kind of logic is difficult to build in GaAs because of the restricted logic forms. These problems were circumvented using the direct mapped scheme shown in Figure 6.8.

In the traditional scheme, renaming is done in the reorder buffer. When a new instruc-

tion is issued, the reorder buffer is examined to see whether it contains the most recent version of any needed operand. The register address for each of the input operands is compared to all of the valid entries in the reorder buffer. A match indicates that the most recent value exists in the reorder buffer. The matching entry is encoded to identify the reorder buffer slot number. The possibility of multiple mappings for a given register complicates the encoding logic.

In the direct-mapped scheme, register renaming is performed in the scoreboard. Each register has a set of bits that indicate the remapping status. To determine whether a register's most recent value is contained in the reorder buffer, the status bits are selected using a 32-to-1 multiplexor. If the mapping is valid, then the map field points to the reorder buffer slot number. The map fields are updated when new instructions are issued, so the mapping automatically points to the most recent renaming value.

Register Retirement Policy

Data for arithmetic and control flow instructions are ready when these instructions are written into the reorder buffer. Long-latency instructions such as loads, multiply, and floating point move instructions must wait many cycles before their data arrive. Long-latency instructions mark the respective reorder buffer entry as not ready; the arrival of the data clears the ready flag and allows the data to be committed to the register files if there have been no exceptions.

Normally, only one instruction result is committed to the register file each cycle. Special logic takes advantage of a common special case. Many instructions do not produce results; jump, branch, and store instructions have no results to commit to the register file. If the destination register for the next entry is zero, no result need be written for the next entry. In this case, the head entry is committed and the head pointer is advanced two locations, skipping the entry with the destination register of zero.

6.5 Load/Store Unit

External Pipelined Data Cache

To accommodate a large enough data cache for good system level performance, the cache RAMs must be external to the CPU. The system supports external caches of 8K-, 32K-, and 64K-bytes. The external data cache is direct mapped, pipelined, and has a three-cycle access latency. Although the latency is three cycles, a new access can begin each cycle. The cache is accessed over two unidirectional 64-bit data busses and a 16-bit address bus.

Nonblocking Loads

To amortize the long primary cache miss penalty, the processor supports multiple outstanding cache misses. As long as the target register of a load instruction is not referenced, other instructions can issue, execute and complete, leaving their results in the reorder buffer. Even more importantly, the access delay of multiple cache misses can be overlapped. Several Miss Service Holding Registers (MSHRs) maintain the state of pending cache misses. An MSHR is reserved for each memory instruction active in the LSU pipeline, and if no MSHR is available, the processor stalls until one becomes free. A machine with only one MSHR cannot overlap memory operations, and must process each load or store instruction sequentially.

Write Cache

The LSU includes a 32-word coalescing write buffer called the Write Cache [Jouppi91]. The 32 data words are organized as four cache lines of eight words per line. The four lines are fully associative. The write cache groups multiple memory references into a single BIU transaction. Memory write behavior has two characteristics that are effectively exploited by the write cache. Multiple writes will often occur to the same address, as would happen in an inner loop during the updating of the loop index. After the first write to the index address, subsequent writes would hit in the write cache, replacing the previous value. Thus fewer BIU transactions are required to keep the memory system coherent. The second memory access pattern that benefits from a write cache is vector-like operations such as

Write Cache Size	micro-TLB hit rates
2 lines	60.2
4 lines	71.5
8 lines	79.5
16 lines	84.5

Table 6.1 SPECint Write Cache MicroTLB Hit Rates

memory copies or floating point intensive code. In these operations, each entry in the write cache is written in succession, but only one BIU transaction is needed to retire the eight words.

Write Validation

Because the MMU is off-chip, the processor cannot retire store instructions until it is known that the address can be accessed without causing a memory fault. Receiving a response from the MMU requires many clock cycles, so simply querying the MMU about each write address is an unacceptable solution. Instead, the write cache divides the address tags into page and offset fields. If the page field of the current write address matches any of the valid page fields contained in the write cache, then write or access faults are not possible. In effect, the write cache operates as a four entry micro-TLB. Table 6.1 shows the micro-TLB hit rate for write caches of different sizes.

Floating Point Support

In this architecture, floating point memory instructions transfer data directly between memory and the integer and floating point register files. To meet package pin constraints, all floating point instruction and data transfers occur via the input and output busses of the primary data cache. Unlike integer store operations, floating point stores do not have data immediately ready at the time the instruction is transferred to the LSU. Thus, write cache eviction and data cache writeback must wait for the data. This adds a certain degree of complexity to the synchronization of the floating point data and the cache line within the write cache.

6.6 Prefetch Unit

A set of hardware prefetch buffers is included to minimize long memory system latencies caused by the fast clock rate and multiple-chip partitioning. The prefetch buffers predict future memory requests and bring the data on chip before it is referenced by the IPU.

Jouppi proposed the addition of a small set of associative prefetch buffers, called stream buffers, to fetch sequential lines ahead of the current program counter [Jouppi90]. The stream buffer consists of a tag register, a tag comparator, a set of status bits and several prefetch cache lines for instructions and data. If a memory reference misses in the primary cache, the stream buffers are checked to see whether the required data have already been requested. Jouppi showed that stream buffers can be highly effective for small caches, reducing the cache miss rate by up to half. Since we have limited space for on-chip caches, these are an ideal solution.

On each instruction or data cache miss, a stream buffer is allocated and initialized to fetch the next sequential line. This buffer initially fetches only a single line. If a subsequent request hits in a prefetch buffer, additional sequential lines are fetched until the buffer is filled.

Stream Buffers

The initial design had four stream buffers each for the instruction and data streams. Trace driven simulations showed large numbers of stream buffers did not improve prefetch hit rates, so the instruction and data stream buffers were merged into a unified pool of four stream buffers.

Several different prefetch algorithms were evaluated. A fetch on miss protocol was adopted. After a cache miss has been detected on chip, the address is sent to the PFU. If the requested address is contained in any of the stream buffers, the data is immediately returned to the requesting unit, and the request does not go off-chip. If the requested address is not in any of the stream buffers, the address is passed to the BIU to fetch from the secondary memory system. The address for the next sequential cache line is calculated, loaded into

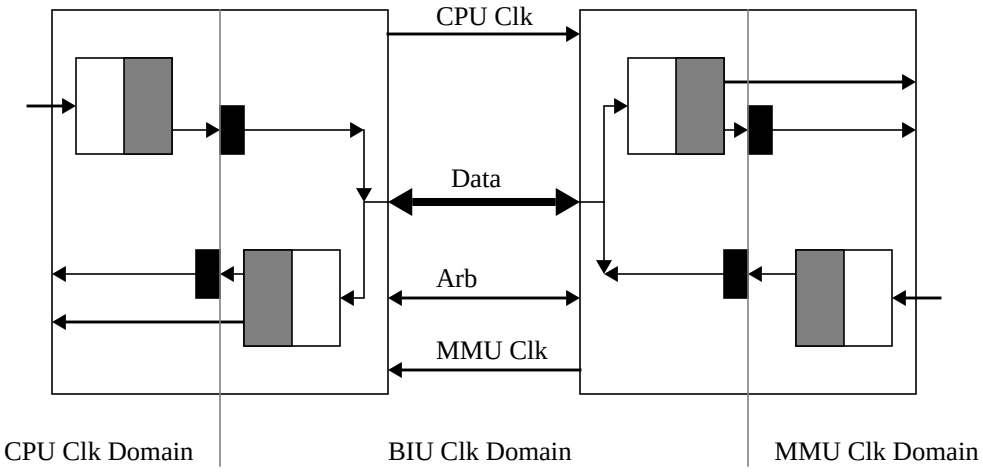

Figure 6.9 BIU Interface

one of the stream buffers, and sent to the BIU as a prefetch request.

6.7 Bus Interface Unit

High-frequency I/O signals were crucial to ensure high bandwidth was available throughout the design, preventing bottlenecks and memory starvation. At the interesting clock frequencies of 200 to 400MHz, transmission line effects dominate off-chip signaling behavior. All processor busses on and off-chip must be terminated. In addition, the busses must be point-to-point. The IPU is connected to the MMU by a unique bidirectional 32-bit bus. The interconnection of the two chips is shown in Figure 6.9. To help the system tolerate the transaction latency, multiple pending requests are buffered in separate transmit and receive queues.

Conventional clocked bus interfaces suffer a bandwidth requirement mismatch between the data lines and the clock lines; data lines have one transition each clock while the clock line has two transitions each clock. Packaging cost is directly proportional to the maximum I/O bandwidth supported. A package designed to support the clock bandwidth requirement on all signal pins is over-designed. The extra bandwidth available on the data pins is wasted.

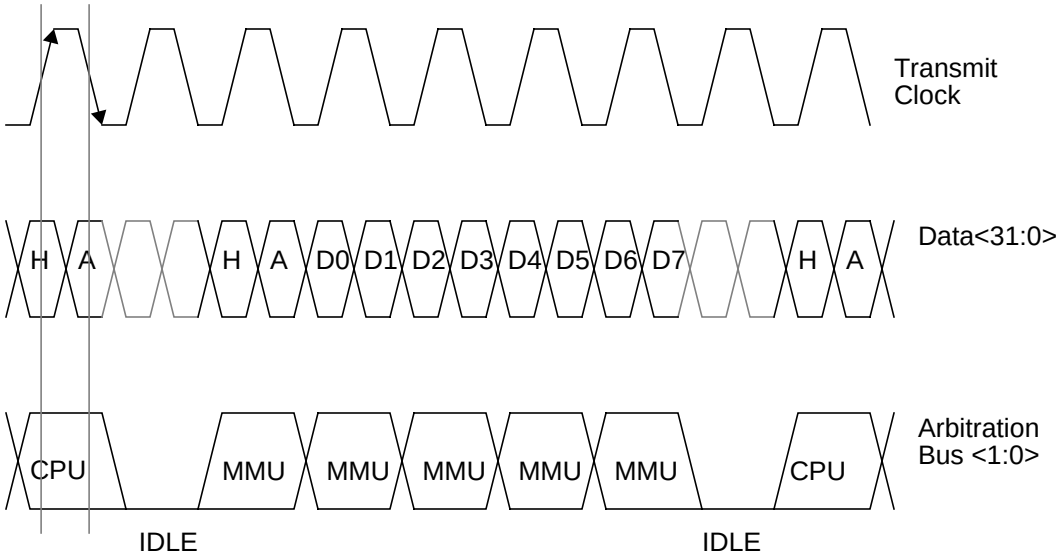

Figure 6.10 BIU Timing Diagram

The round-trip latency to main memory in this system can be quite long, as many as 100 cycles. This long latency precludes using a tenured bus that locks the bus for the duration of each transaction. Instead, a split transaction bus is used. Each transaction is divided into two individual data transfers, a request and a response. Each transfer uses a packet scheme consisting of two 32-bit header words and 8 optional data words. The two header words contain a 32-bit command field, and a 32-bit address field.

A signaling scheme that transfers data on both clock edges has been adopted. This equalizes the bandwidth requirement for all signals, and allows data transmission to occur at the maximum rate the package and interconnect medium will allow. The signaling scheme is shown in Figure 6.10, and is similar to that used by RAMbus for high-bandwidth DRAM communication [Kushiyama93]. Data is transmitted on each edge of the clock, and the data is shifted by 90 degrees with respect to the clock to give the maximum setup and hold times possible. Clock skew remains a critical factor, but by routing the clock and data lines together, the skew can be minimized. Running the BIU bus at twice the IPU frequency gives the same peak bandwidth as provided by the 64 bit data cache interface.

A distributed arbitration scheme is used to gain access to the bus. The BIU interface

uses a carrier-sense distributed arbitration scheme. A two-bit arbitration bus connects the IPU to the MMU using wire-OR Gunning Transceiver Logic (GTL) signals [Gunning92]. When the bus is idle, the two-bit arbitration lines are pulled to <00> by the termination resistors. In this state, either unit is free to immediately begin transmitting data. When the IPU sends data, it drives logic <10> onto the arbitration bus. When the MMU sends, it drives <01> onto the bus. As soon as the receive unit recognizes that the bus is busy, it inhibits its own transmissions until the bus becomes free.

It is possible for both units to see the bus is free and try sending data simultaneously. In this case, the wire-OR arbitration signals are driven to <11> and a true collision occurs. In case of a collision, all partially received bursts are discarded and the complete burst is retransmitted. Because the IPU initiates most transactions, it is usually waiting for some data from the MMU. In case of a collision, the MMU is given priority on the bus, giving it the opportunity to return data to the IPU.

The transmit and receive queues serve many functions. While the bus is busy, multiple requests can accumulate in the transmit queue. When the bus becomes free, all pending transactions can be sent in one burst. The finite size of the queues introduces the possibility of a chip sending more data than there is free space in the receiving queue. This overflow condition is detected, and the error is handled in the same way as a true collision. If the receive queue fills, the receiver forces a <11> onto the arbitration bus. To the transmitter, it looks as though a collision occurred, and the entire burst is retransmitted. The transmitter does not wait for the receive queue to drain, but instead re-sends the data as soon as the bus clears after the fifo-full collision. Because the receive queue is larger than the transmit queue, there is usually space available soon after a collision.

The CPU devotes nearly all of the off-chip bandwidth to the management of the primary data cache. Of the 256 signal pins on the CPU, 180 are used for reading and writing the data cache.

6.8 Architectural Evaluation

Three machine models, called the large, baseline and small models, were chosen to evaluate the effects of resource allocation and memory latency. The hardware resources allocated to these models are listed in Table 6.2. Each of the three models is evaluated with one or two execution pipes, and with secondary memory system average latencies of 17 and 35 cycles, corresponding to medium and fast clock rates. Thus, results for 12 configurations are reported.

Mulder's register bit equivalent (RBE) model was used to evaluate the implementation cost in chip area of the different configurations [Mulder91]. The RBE model provides a normalized measure for the area cost of different microarchitectural components. For our machines, the RBE is the area required to implement a 1-bit static latch. In GaAs DCFL, one static latch requires 16 transistors and occupies an area of about 3600 square microns. Static RAM elements are denser, with a single bit requiring an area equal to 0.5 RBE. However, RAM blocks have additional overhead devoted to decoding and sensing. This overhead is a significant fraction of the total area of small RAM blocks. The cost of each microarchitectural element is listed in Table 6.3. These figures are based on layouts obtained during chip design. In addition to memory elements, the cost of an execution pipeline is also estimated. An important assumption in this analysis is that the cost of interconnecting microarchitectural elements is an overhead that scales with the sum of their areas.

The cost of data cache is not included in this analysis because die-size limits forced the data cache off chip. A more comprehensive cost analysis would optimize a system consisting of the IPU, FPU and data cache chips on the MCM in a way analogous to the intra-chip

Model	I Cache Size	D Cache Size	Write Cache Size	Reorder Buffer Entries	Prefetch Buffers	MSHR Entries
Small	1 K-byte	16 K-byte	2 lines	2	2	1
Baseline	2 K-byte	32 K-byte	4 lines	6	4	2
Large	4 K-byte	64 K-byte	8 lines	8	8	4

Table 6.2 The Three Machine Models and Their Associated Resources

IPU Element	Cost in RBE
1 K-byte Cache Block	8,000
2 K-byte Cache Block	12,000
4 K-byte Cache Block	20,000
1 Write Cache Line	320
1 Prefetch Line	320
1 Reorder Buffer Entry	200
1 MSHR Entry	50
1 Integer Execution Pipeline	8192

Table 6.3 Processor Element Cost in RBE Units

design problems considered in this chapter. Finally, it is important to note that increases in area will slow the clock cycle [Olukotun92]. This effect is not included in the model presented here.

6.9 Study Results

The size of on-chip memory structures and the number of execution pipelines were varied to study the trade-offs between memory structures and execution logic. The Cycles Per Instruction (CPI) for each benchmark and processor configuration were measured, and can be compared to the implementation cost for each configuration to evaluate the effectiveness of various memory and pipeline combinations. The base model instruction cache hit rate is 96.5% and data cache hit rate is 95.4%; these numbers agree with previously published results [Gee93].

Dual Issue Percentage

Although a restrictive instruction-pair based dual-issue model is used, a large percentage of instructions are issued in pairs. Table 6.5 shows the percentage of instructions that issue as part of a dual issue pair.

Benchmark	espresso	li	eqntott	compress	sc	gcc
Dual Issue Percentage	58.5	54.3	65.1	65.7	62.8	58.7

Table 6.4 Dual Issue Frequency

Model Performance Evaluation

Figure 6.11 shows the results for dual and single issue performance for the 3 baseline models with average secondary cache latencies of 17 cycles. Two graphs are shown in the figure, one for the single issue performance and the other for the dual issue performance. The six integer SPEC benchmarks are used to evaluate performance. A vertical line for each machine configuration connects the minimum and maximum CPI values for the set of benchmarks, the single issue model has circular endpoints and the dual issue model has square endpoints. The average CPI value for each machine configuration is connected by a sloping horizontal line, one line connects the dual issue configurations and the other con-

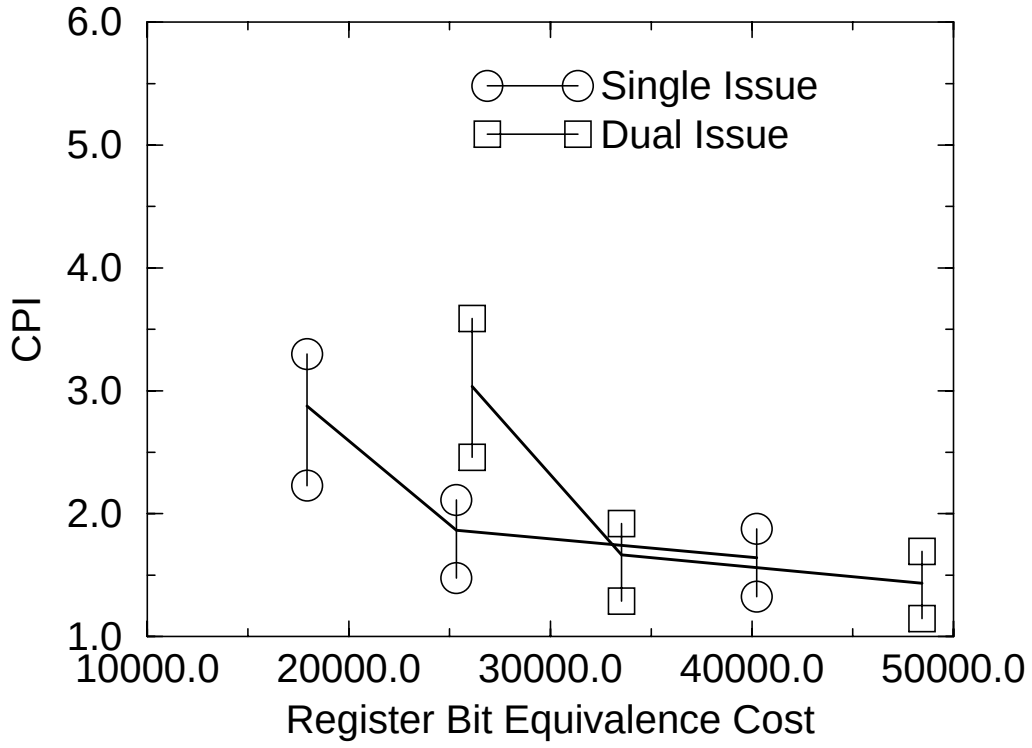

Figure 6.11 Dual and Single Issue Performance, 17 Cycle Memory Latency

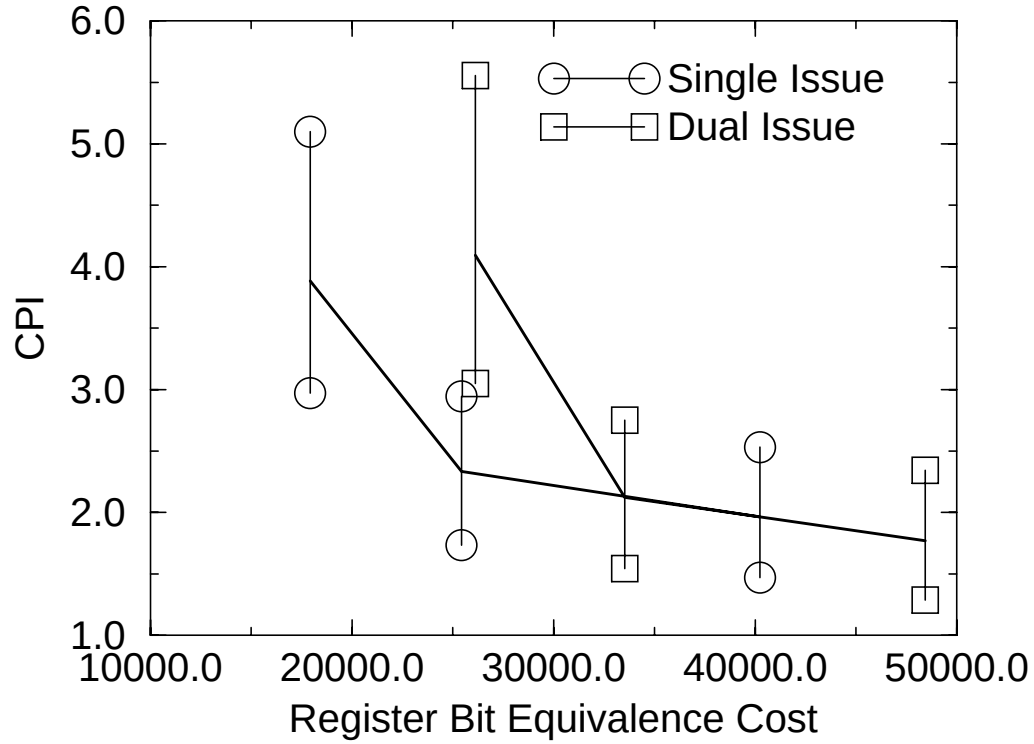

Figure 6.12 Dual and Single Issue Performance, 35 Cycle Memory Latency

nects the single issue configurations.

For each graph the cost difference between the two curves is caused by the addition of a second execution pipeline (8192 RBEs). With 17-cycle latencies, the addition of the second pipe results in higher average performance with the baseline and large models. The single issue base model has a similar cost and much better performance than the dual-issue small model. The large model with dual issue achieves the best performance by 12.7%, but with a hardware cost increase of 20.4%. With 35-cycle secondary latencies, shown in Figure 6.12, the dual and single issue machines have cost-performance curves similar to those of the 17-cycle machines; the large dual-issue model achieves a 9.9% CPI improvement over the single issue model when maximum resources are used.

Instruction and Data Prefetching

Table 6.5 and Table 6.6 show the prefetch buffer hit rates for the instruction and data streams. A prefetch hit occurs if the data misses in the primary cache and hits in one of the

model	espresso	li	eqntott	compress	sc	gcc
small	56.26	50.79	94.89	50.73	45.97	58.89
baseline	61.02	45.33	88.34	53.13	49.01	57.75
large	57.33	40.56	51.41	53.75	48.05	56.10

Table 6.5 Integer I Prefetch Hit Rate Percentages

model	espresso	li	eqntott	compress	sc	gcc
small	7.06	9.80	2.85	8.33	22.34	7.84
baseline	8.95	14.41	2.29	13.13	27.42	8.63
large	7.82	14.57	1.53	17.16	30.00	10.02

Table 6.6 Integer D Prefetch Hit Rate Percentages

prefetch stream buffers. Average hit rates for the integer SPEC benchmarks are 58% for the instruction stream and about 12% for the data stream. Floating-point prefetch data hit rates average 14.4% for the base model, but the peak hit rate for some of the benchmarks is as high as 60%.

Figure 6.13 shows the effects of removing the prefetch buffers from the 17-cycle memory latency dual issue models. Prefetching is of little benefit in the small model for many reasons. There are only two buffers in the small model, which leads to thrashing between instruction and data references. In addition, the cache miss rates are high, leaving little spare bandwidth for prefetching.

Prefetching is much more successful in the base model, resulting in an average improvement of 11% for the 17-cycle latency system. Figure 6.14 shows that prefetching helps even more in the 35-cycle latency system, improving performance for the base model by 19%. The large model also sees a significant improvement: 11% for the 17-cycle latency and 17% for the 35-cycle latency system. In addition to the average performance, prefetching substantially improves worst case performance, reducing the CPI for the short-latency case in the study by 25% and in the long latency case by 35%. The cost of adding prefetch buffers is comparatively small; for the baseline configuration, the prefetch buffers are only 20% of the instruction cache size.

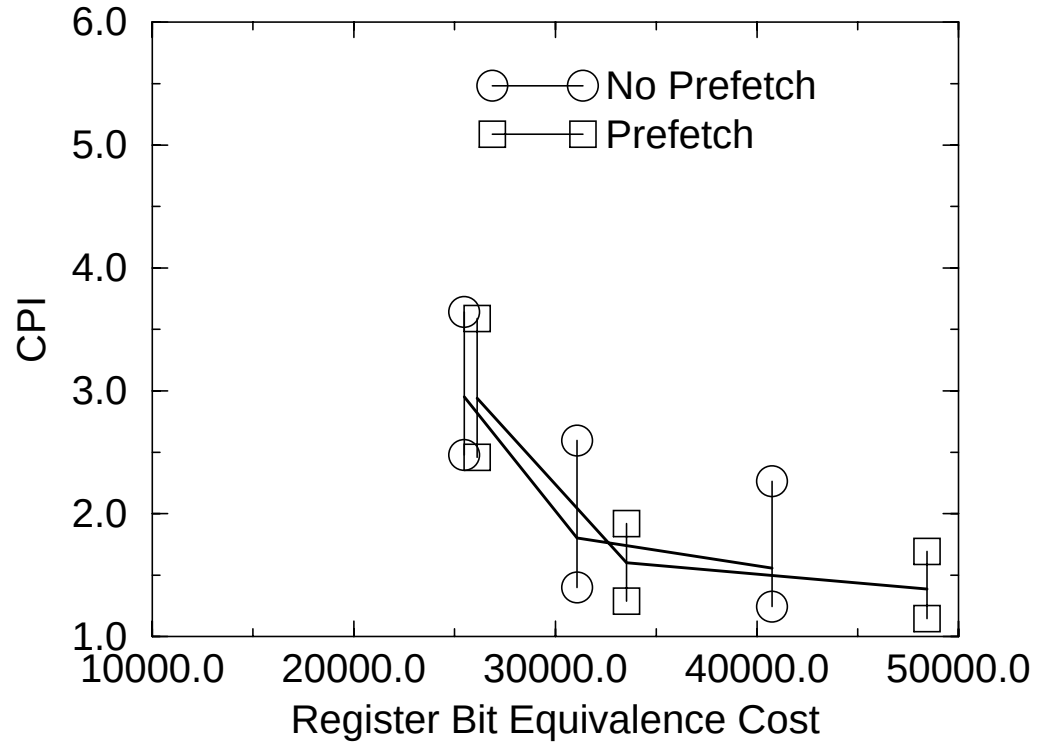

Figure 6.13 Effects of Prefetch Removal, 17-cycle Memory Latency, Dual Issue

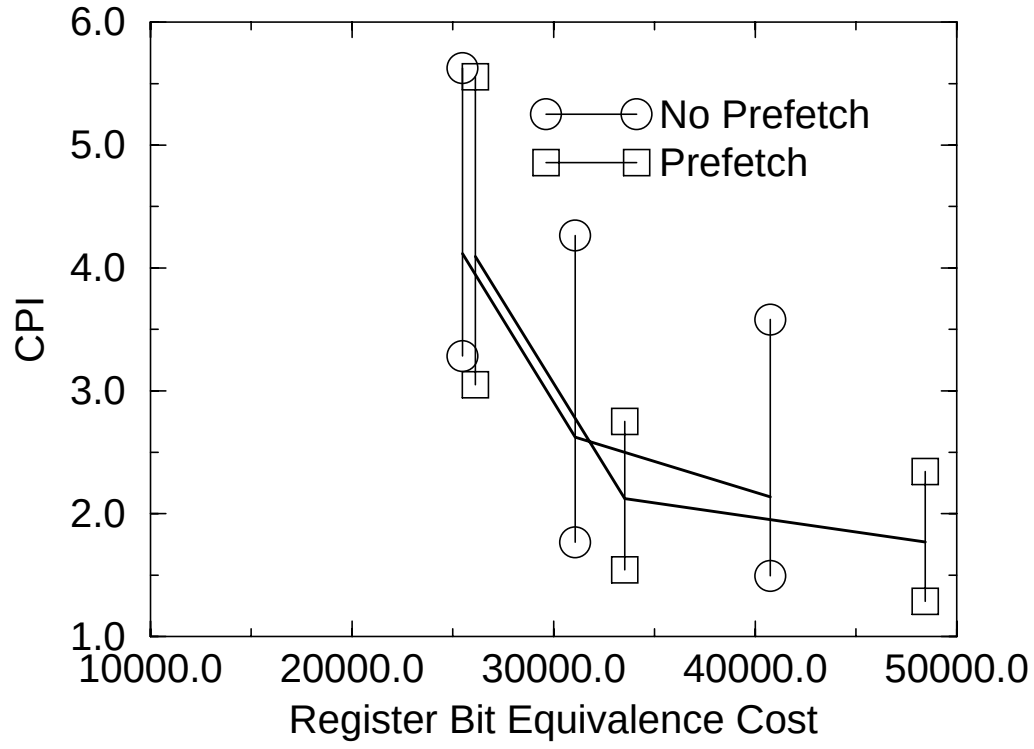

Figure 6.14 Effects of Prefetch Removal, 35-cycle Memory Latency, Dual Issue

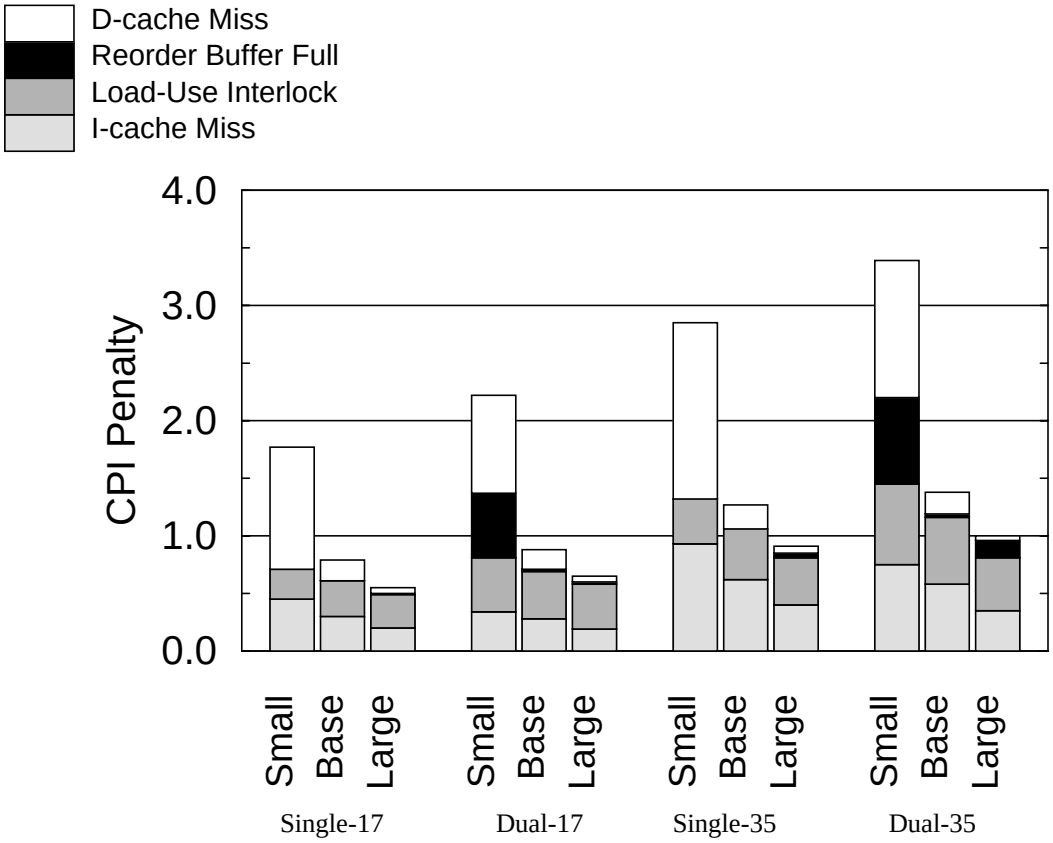

Figure 6.15 CPI Losses to Different Causes

Execution Unit Stall Distribution

The IPU has 4 major stall conditions: instruction cache stall while waiting for instructions, load stall when the result of a load instruction is referenced before it has been returned by the LSU, Reorder Buffer full stall, and LSU stall when the LSU is full or is using the data busses to fill the cache.Figure 6.15 shows the CPI penalty contributed by each of these stall conditions.

Except for the small model case, most stalls are caused by instruction misses and data cache accesses. In the small model, most cycles are spent waiting for data from the LSU. In the base and large models, performance is not very sensitive to the size of the IPU reorder buffer because the processor often stalls when it references the result of a load instruction before the reorder buffer fills. In the large model, the small percentage of LSU-Busy stalls indicates that most of the data hits in the cache; the large percentage of load-use interlock stalls is caused by the three-cycle latency of the pipelined data cache.

Degree of Non-blocking Data Cache

One of the largest contributors to high performance is the ability to support multiple outstanding load instructions simultaneously. The number of outstanding data cache misses is set by the number of MSHR registers. Figure 6.16 shows the effect of changing the number of possible outstanding memory references for the different processor models. The graph with the circular endpoints shows the performance of the 3 models from Table 6.2, with 1, 2, and 4 MSHR entries. The graph with the square endpoints changes the number of MSHR entries in each model to 4, 4 and 2 for the small, baseline and large models, respectively. The small model shows a dramatic performance increase when additional MSHR registers are allowed, and the base model shows a small improvement when adding two MSHRs. Note that the additional MSHR entries in the small model greatly reduce the variance in CPI for different benchmarks, indicated by the size of the vertical line connecting the minimum and maximum CPI values. The large model shows some performance decrease when the number of MSHR registers are reduced from four to two. All models get highest performance when 4 MSHR entries are available.

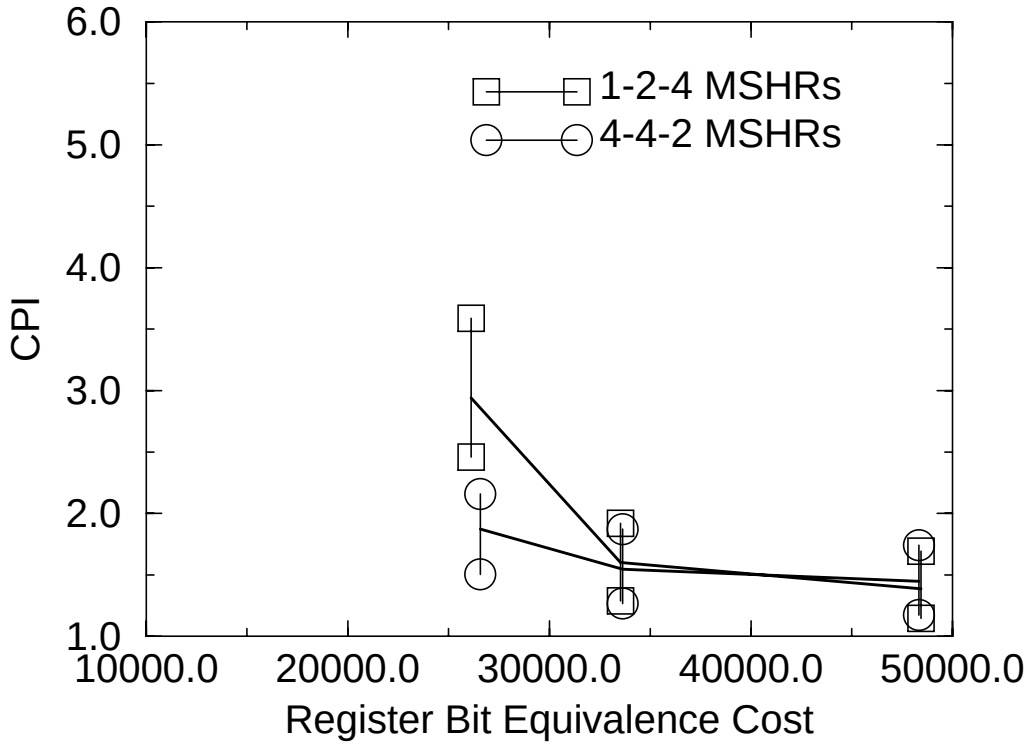

Figure 6.16 Effects of Changing MSHR Count on Dual Issue Model

model	espresso	li	eqntott	compress	sc	gcc
small	29.22	35.97	31.84	37.82	46.61	42.82
baseline	37.17	49.17	48.34	46.29	52.53	54.93
large	43.73	60.52	60.03	59.87	59.56	63.17

Table 6.7 Integer Write Cache Hit Rate Percentage

Write Cache Size

Table 6.7 gives hit-rate statistics for the on-chip write cache of the different models. The write cache size varies from two lines in the small model to eight lines in the large model. The hit rate includes both load and store data accesses. The write cache has a surprisingly high hit rate, even for the small model. This high hit rate helps to reduce significantly the write traffic off chip. The number of store transactions is reduced to 44% of the number of store instructions that would otherwise be seen for the small case, to 30% for the base model, and to only 22% for the large model. This is more than a two-fold reduction in write traffic for the small model and nearly a five-fold reduction for the large model.

Analysis of Integer Data and Recommendations

Figure 6.17 presents all of the simulation data points for the latency-17 data sets on the espresso benchmark. The other benchmarks and latencies give similar performance curves. This graph demonstrates the trade-offs between cost and performance for our machine model. The points labeled A all have only a single MSHR and lie well above other configurations of equivalent cases, showing the negative impact of blocking caches. The points labeled B correspond to the large model. A performance plateau exists here that yields little gain in performance even if significant additional cost is expended. The benefits of prefetching are demonstrated by comparing points C and D. They differ only in that D adds prefetching. In general, the small model shows a large performance increase with additional resources, the base model shows a small increase, and the large model shows negligible improvement.

The previous data suggest some obvious changes to our baseline machines. All models benefit from increased MSHR entries; because the cost is low, adding MSHR entries pro-

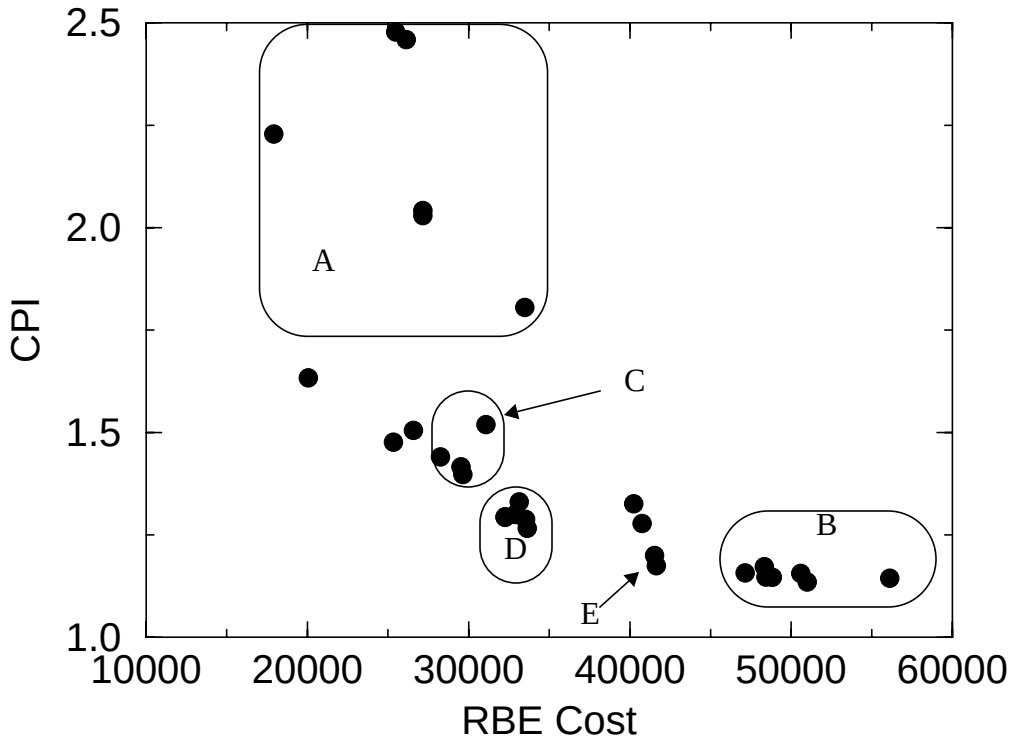

Figure 6.17 Espresso Full Data Set, 17-cycle Memory Latency

vides a price-performance benefit. A write cache larger than in the baseline model has little performance benefit. In addition, the prefetch performance of the large model is no better than that of the baseline model. The point labeled E on the graph represents a model that reduces the resources for the large model to a 4 K-byte I-cache, a 4-entry write cache, a 6-entry reorder buffer and 4 MSHR entries. This reduced machine uses significantly fewer resources than the machine configurations identified with the points labeled B, with only slightly reduced performance. Dual issue is reasonable only if supported by sufficient memory resources and becomes less attractive as memory latency increases.

6.10 Summary

This chapter presented the area/performance tradeoffs for E/D MESFET GaAs microprocessors. Advanced memory architectural ideas such as nonblocking loads and prefetching were found to increase performance on machines with both 17-and 35-cycle secondary memory latencies. Dual instruction issue had the greatest performance benefit with lower

memory latencies, indicating that, for fixed access-time memory, it is more difficult for faster clock rate machines to make use of instruction level parallelism.

CHAPTER 7

The Design Process and Verification

7.1 Overview

The previous chapter described the microarchitecture of the Aurora III processor, and the effects of different resource allocations on processor performance. This chapter discusses the design process, how architectural and design decisions interact, and what can be done to produce better designs more efficiently in the future.

7.2 Critical Path Optimization

The modular design style adopted to partition the integer processor chip into five major blocks simplified the design, but greatly increased the effort required to minimize the critical logic path delays to the required levels. Kunkel and Smith studied the effects of logic depth on vector processor performance [Kunkel86]. They found a depth of eight levels per clock cycle was the optimum value for the Livermore Loop kernels [McMahon72]. Our paths were longer because of the increased number of levels caused by the use of only NOR gates. To reach the performance target of 330 MHz outlined in Chapter 4, critical path lengths of no more 15 gates were needed for each clock phase.

Table 7.1 shows the critical path length in each of the major functional units at the start

Module	Initial Path Length
IFU	24
IEU	28
PFU	35
LSU	50

Table 7.1 Initial Critical Path Lengths

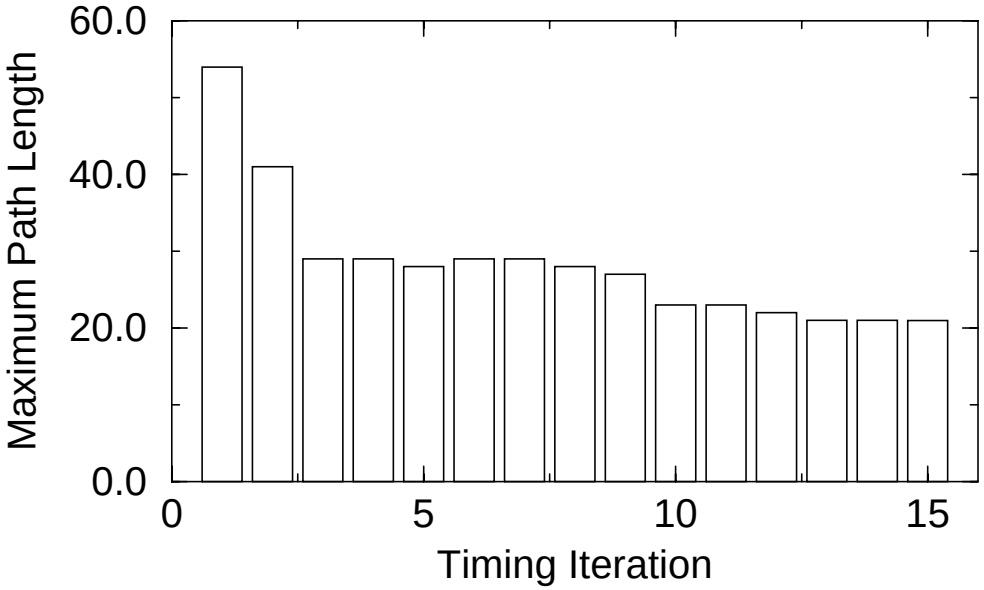

Figure 7.1 Critical Path Length for Aurora III CPU

of critical path optimization. It was immediately clear that the initial performance targets for the chip would not be met. A revised minimum performance goal of 200MHz was adopted, and work began on critical path reduction. Each of the functional units was optimized to reduce the longest paths in each module to the revised target logic depth of 24 levels.

Full chip timing optimization was then begun, concentrating on the intermodule paths. The logic paths between modules were poorly optimized in the initial design, and also required re-design work. Initially, the longest path was 53 gates. Figure 7.1 shows the improvement in critical-path length after several optimization passes. After about six iterations, further progress was limited by a long path between the IEU and the LSU. It was ultimately necessary to insert an additional pipeline stage between the IEU and LSU to allow sufficient time for the generation of the LSU control signals.

The additional pipeline stage allowed for rapid progress, and all paths were ultimately reduced to 21 gates or less in length. Figure 7.2 shows the total number of paths longer than 18 gates for each timing iteration. Even when an iteration did not decrease the maximum path length, the number of long paths could be substantially reduced.

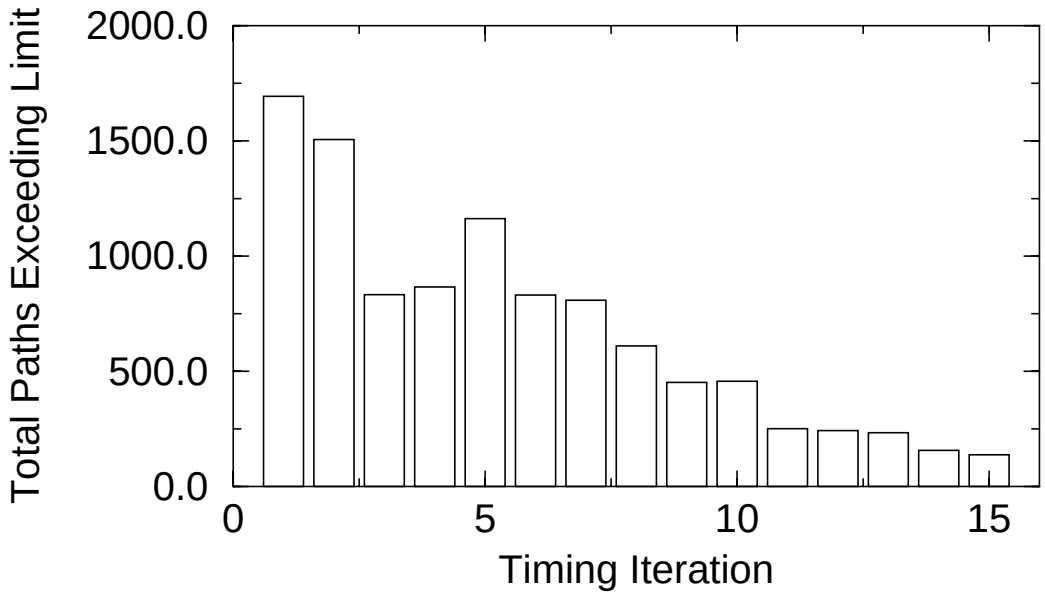

Figure 7.2 Total Number of Critical Paths Greater Than 18 Levels

Two Phase Clocking

The Aurora III processor used the same clocking scheme as the Aurora II processor had used, as shown in Figure 7.3. More effort was devoted to balancing critical path lengths between the two phases than in previous Aurora processors. Because level-sensitive latches are used, time can be borrowed from either the previous or the following phase. CAD tools were developed that measure the amount of borrowing possible. Borrowing time from the previous phase resulted in a total maximum critical path length of 38 gates for the sum of two phases. If time borrowing were not used, the total critical path for two phases would have been 42 gates. Time borrowing resulted in nearly a 10% improvement in cycle time. The final critical path of 38 gates results in an estimated clock cycle of 260 MHz, midway

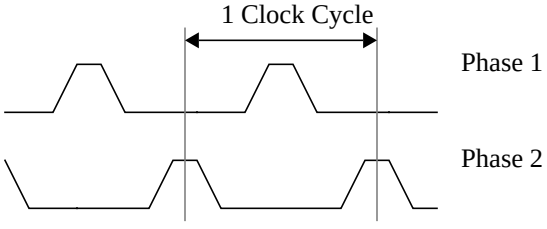

Figure 7.3 Two Phase Clock Diagram

between the 200 MHz minimum and the original 330 Mhz target.

7.3 Processor Verification

Experience with the Aurora II processor showed that more concern with verification was needed from the beginning of the design process. Early effort in testing often produces large gains later in the project.

As increased resources have allowed higher computational throughput, both through pipelining and parallel operation, systems have become more complicated and therefore harder to test and verify. The number of possible interactions between operations grows exponentially as the instruction issue width increases. Scan testing can handle the problem of manufacturing test, but the verification problem, ensuring that a design meets its specification, is still an area of active research.

Pseudo-random testing has emerged as a powerful technique for exercising complex systems. Several recent reports have described the use of pseudo-random vector generation to test complex, interacting state machines in microprocessor systems [Diodato85, Shalem87, Wood89, Harper91, Anderson92, Gupta94]. This approach involves generating a random test vector set and an expected result. The test script is applied to the model and the resulting final machine state is compared to the expected value. Any discrepancy is flagged as an error.

A verification system was built for the Aurora I processor using the **`ptrace()`** system call to compare the contents of the simulated CPU registers to the contents of an actual machine running the same program [Nagle91]. The actual machine and the simulation model were each stepped one instruction, and the contents of the register files were compared. The main difficulty with this method was that the **`ptrace()`** call gave the state as though the machine was unpipelined, and the simulation model had a 5 stage pipeline, so some post-processing was required to properly align the results after memory and control-flow instructions.

The superscalar nature of the Aurora III processing units further complicates the veri-

fication. The IPU consists of 5 major elements that operate independently and concurrently. The queues and reorder buffer make it difficult to determine the exact architectural state of the machine at each clock cycle. In addition, the FPU has its own instruction and data queues, and may be processing instructions significantly out of sequence with those in the IPU. Because the reorder buffer, LSU, and the queues distribute the architectural state throughout the machine, it is difficult to compare the state of the model with that of a real machine without letting all the queues drain, thus changing the behavior of the test.

Pseudo-Random Testing Methodology

A verification methodology built around self-checking pseudo-random programs was developed to verify the Aurora III processor. The programs check for the expected result continuously as the program progresses. Self-checking programs are well known in compiler research, as compiler writers have a similar verification task [Bird83]. Both assembly language and C language random test programs are generated. The assembly random tests are constructed from two primary sources: a library of small known-good single-instruction tests, and larger memory and control-flow tests that are created on the fly. The C language tests are all generated on the fly, and then linked with a random selection of the assembly level tests to produce a final test program.

The test generators for Aurora III are written in the Perl language [Wall91]. The memory system test is the most rigorous, and has been responsible for detecting the most errors. The memory system test consists of a randomly generated data section, a sequence of memory operations, and a final exhaustive state verification. The data section is of random size and starts at a random word address. The memory operations consist of a random sequence of writes, block copies and tests. A write operation generates a random value and stores it to a random address. A block copy reads a sequence of data from one random address and writes it to another, and a check reads a random value and compares it to the value that should be at that location.

The other assembly language test that is generated on the fly is a control flow test. A sequence of random jumps, branches and calls is constructed in random order. The target

of each branch verifies that it was reached from the proper control flow instruction.

During instruction execution, the source for an operation changes as the operation flows down the pipeline. Possible source locations include the register file, a forwarding path from either of the execution pipelines, the Load/Store Unit or the reorder buffer. To exercise all possible forwarding paths, a NOP insertion phase adds NOP instructions at a random rate to the final test program. These NOPs do not affect the computed results, but vary the interlocks and forwarding paths. Two types of NOP instructions are used: computational NOPs and control flow NOPs. Computational NOPs perform a null operation, such as adding 0 to a register or ANDing a register with itself. Control flow NOPs are added to stress the branch prediction and speculative execution circuitry. These instructions typically branch if a register is not equal to itself, and cause branch predicted misses at a selectable rate. When an error is detected, the test is immediately halted. The test case and test log file are archived for later analysis. Halting the test immediately on the first error detected helps to isolate the fault to a small code region, and simplifies the diagnosis of the fault.

Testing Environment

It was originally believed that a hardware emulation environment would be needed to adequately test the full Aurora III CPU model. The introduction of a compiled Verilog simulator greatly improved the simulation throughput, reducing the incentive to port the design to the hardware emulator.

An automatic test environment was developed to utilize the large compute capability available in the University of Michigan Computer Aided Engineering Network (UM-CAEN). A set of automatic scripts continuously generated random tests and farmed them out over the network. Tests were run on five to ten workstations constantly for over a year. The limit of ten concurrent simulations was set by the number of available simulator licenses.

Figure 7.4 shows a graph of the cumulative verification cycles for the duration of the testing process. Simulations were initially run on only one workstation, so the number of cycles increased slowly until the 100-day mark was reached. The number of simulation cy-

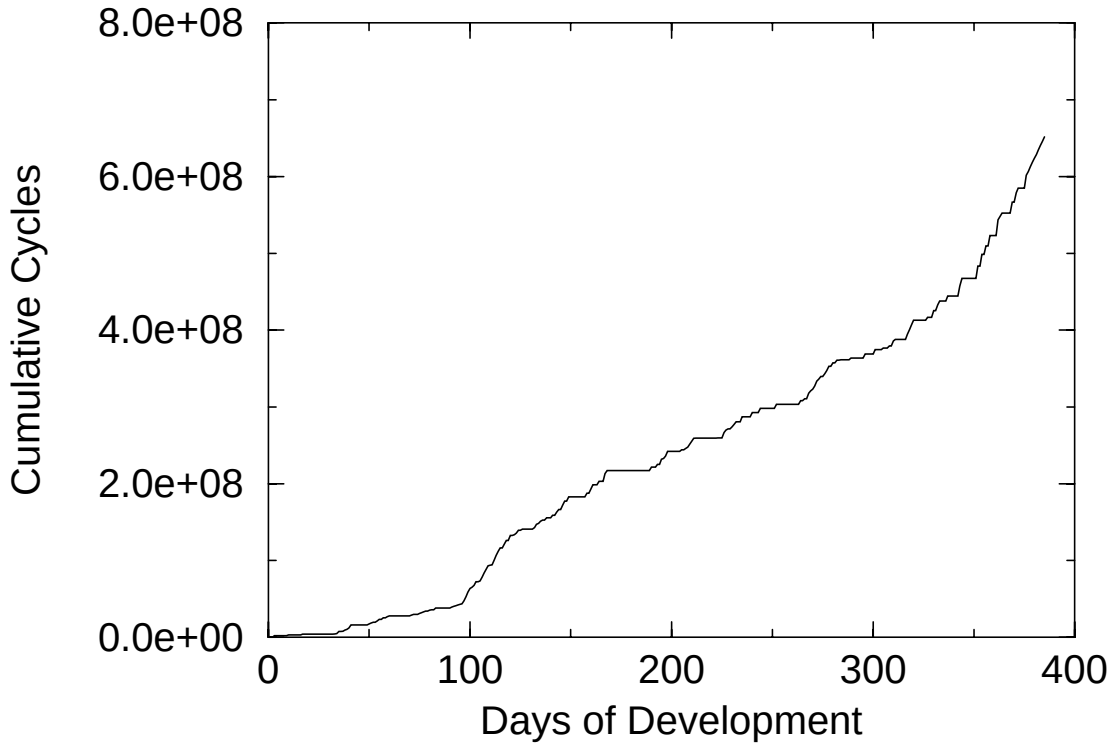

Figure 7.4 Total Cumulative Verification Cycles

cles accelerated rapidly when the parallel network testing system became available at around day 100. There was about one month of rapid increase, then the rate slowed somewhat when the FPU testing began to compete for available resources. At around day 350 the FPU testing was completed and the IPU again utilized all available testing resources. In all, about 650 million simulation cycles were run.

7.4 Scheduling

As in most microprocessor design projects, a concurrent-engineering design flow was adopted. Many of the design tasks were processed in parallel, before the entire list of requirements for the constituent components was known. A serial design flow would have resulted in less total work, and significantly less rework, but would have greatly extended

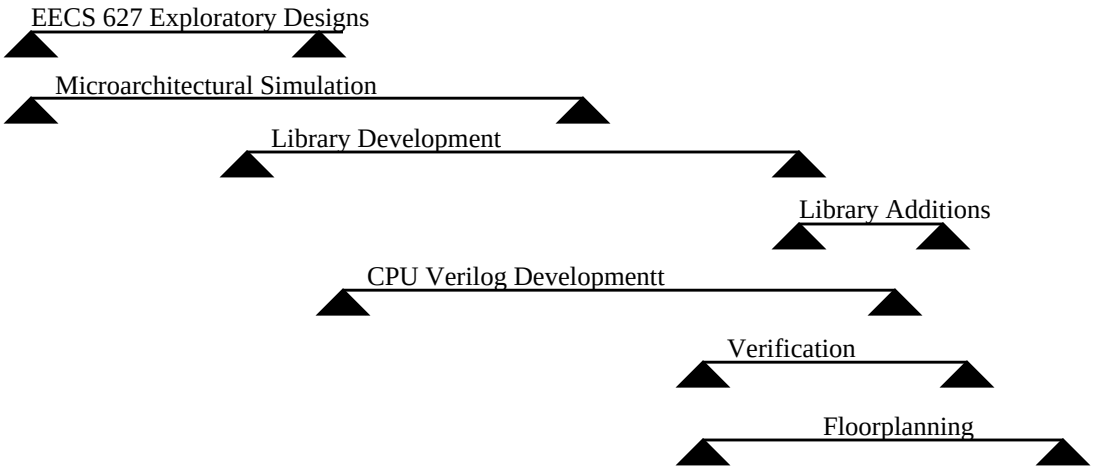

Figure 7.5 Top Level Design Schedule

the already long design schedule. A high-level view of the project schedule and areas of concurrent design is shown in Figure 7.5.

Developing and maintaining an accurate schedule is a crucial element in the successful completion of any complex project. Valuable insights into the development process can be obtained by continually monitoring the project schedule. Figure 7.6 shows a scheduling diagram for Aurora III using a method that was originally developed by Digital Equipment for tracking the progress of large projects in a concise form [Conklin92]. The scheduling diagram presents information in two dimensions. Along the top of the diagram is a quarter-

```
        |Q3 1992 |Q1 1993 |Q2 1993 | Q3 1993 | Q4 1993 |Q1 1994 | Q2 1994| Q3 1994|Q4 1994
        |10|11|12|01|02|03|04|05|06|07|08|09|10|11|12|01|02|03|04|05|06|07|08|09|10|11

10.92   S     D   R B   C       F     T
01.93     S             D R B C     F T
04.93     S                 D   R          B   C     F     T
07.93     S                          D R B   C       F     T
10.93     S                            D R B    C       F     T
01.94     S                            D R     B   C       F     T
04.94     S                            D R     B           E C     F     T
07.94     S                            D R     B               E      P C F     T
10.94     S                            D R     B               E           P    C


        Milestones
                      S: Spec
                      D: Design Complete
                      R: Run first code
                      B: Bug free gate level design
                      E: Bug free gate level design with Exceptions
                      P: Critical Paths Optimized
                      C: Tapeout
                      F: Return From Fab
                      T: Testing complete
```

Figure 7.6 Aurora III Schedule Slippage Diagram

ly and monthly time line. Beneath the time line, the diagram presents a series of schedules, each done at a progressively later date. The diagram consists of a sequence of major milestone schedules, generated at the dates listed in the left most column. A short description of the milestones is listed below the schedule graph. Slippage in the schedule is evident when the milestones for successive schedules are not vertically aligned. Slippage indicates an error in estimating the work required for a task.

A good schedule will contain all the needed milestones in the initial milestone timeline, and the milestones will be completed near the estimated completion dates. The schedule in Figure 7.6 shows evidence of grave problems in the project. As time progresses down the graph vertically, the estimated completion dates for the uncompleted milestones are continually extended. The initial completion date for the D milestone, design complete, was late November 1992. This milestone was finally completed in July 1993, 9 months late and in error by a factor of 5 from the initial time estimate.

It is important to note that this is not bad; this is exactly the purpose of a schedule, to monitor progress and identify problem areas in the development. The personnel involved in the development were inexperienced in project management and schedule development, which led to wildly optimistic schedules. These schedules were derived from examining similar phases of the Aurora II project, but failed to note the effect of two important facts: the Aurora II design was not adequately verified, resulting in a design time that was too short, and the Aurora III design is at least 10-fold more complicated than the Aurora II design. Because of these factors, the initial schedule for the Aurora III processor should have been significantly longer than that required for the Aurora II project, but the initial schedules for the two projects were similar.

The Aurora III development had an additional source of error in the schedule. Two major milestones were not included in the initial schedule, because it was believed that they would not contribute significantly. The first missing milestone was full support for exceptions. Turning on interrupts and exceptions caused a large jump in the rate of bugs detected. The problem was not so much in the logic which implements exceptions, as the fact that enabling interrupts and exception revealed existing bugs in many of the other units and in

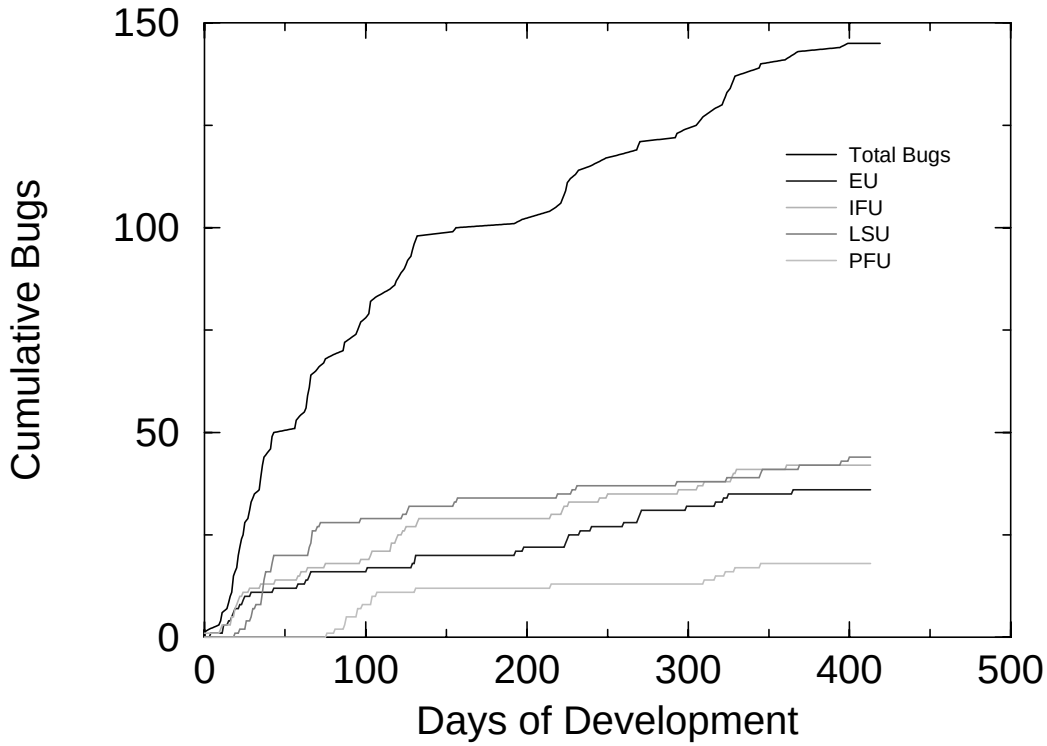

Figure 7.7 Total Development Bugs by Unit

the interactions between units. The second missing milestone was optimization of the critical paths. Because many of the designers of the Aurora III had some experience in high speed logic design, it was believed that the critical paths would be optimized as the design progressed. The designers would intuitively avoid long data and control paths. This turned out to be an incorrect assumption, because the design became so complicated it became impossible to completely understand what impact bug fixes and design enhancements would have on the critical paths. Optimizing the critical paths took nearly five months of unscheduled time.

7.5 Bug Tracking

Throughout the design, bugs were tracked to estimate the progress towards completion. Figure 7.7 shows the progression of total bug count from the start of system integration test-

Module	Total Bugs	Lines/Bug	Statements/ Bug
PFU	18	114	61
IEU	36	214	133
IFU	42	118	47
LSU	45	205	71

Table 7.2 Bug Statistics by Module

ing until the design was frozen for tapeout. The bugs shown do not include those found in unit module testing. System integration testing was begun only when it was believed the individual units were free of major design flaws.

Table 7.2 summarizes the total bugs found per unit, as a function of the size of the Verilog code for each unit. The BIU was completed and debugged before the system integration process for the rest of the CPU, and is not listed. The bug rate for the different modules is quite uniform, whether based on lines or on statements.

Table 7.3 summarizes the design errors based on type, as described below:

Wrong Logic: The logic did not perform the intended function.

Stall: Part of the pipeline stalls when it should not, or does not stall when it should.

Type	number
wrong logic	70
stall	19
handshaking	12
missing logic term	11
sequence	11
phase	10
memory consistency	6
bypass	3
other	3

Table 7.3 Design Errors by Cause

Handshaking: Improper interaction of multiple state machines, often caused by some part of the machine ignoring a queue-full signal.

Missing Logic Term: A signal needed to ensure proper operation was missing from one or more logic gates.

Sequence: Operations were not performed in the proper sequence. For example, re-starting the execution pipeline one cycle after the program counter when servicing a cache miss, causing an instruction to be lost from the pipeline.

Phase: Signals from both clock phases feeding a combinatorial logic block. This causes the combinatorial outputs to change on both edges of the clock, and violates latch setup and hold times.

Memory Consistency: Out-of-date memory values being written over newer versions somewhere in the machine.

Bypass: A forwarding path was not properly enabled, resulting in the wrong data being used for calculations.

Other: Changes made to improve performance.

Of the 145 bugs found, 117 involved single isolated conditions, 26 involved two simultaneous interacting conditions, and two involved three simultaneous conditions. It can be difficult to find errors that result from multiple interacting causes. These cases are where the power of random testing is most useful.

The results of the testing were somewhat surprising. It was expected that most of the errors would be in the LSU and the execution unit. The LSU is the most complicated portion of the processor, responsible for maintaining the coherent state of the memory system and supporting multiple concurrent memory operations. The number of errors found in the LSU was expected. What was not expected was that the IEU would have few major errors, and the IFU would have many more. Although the design is superscalar, very few errors resulted from the data bypassing and forwarding needed to move the data between the reorder buffer, both pipelines and the register files.

The test process was limited by the ability of the design personnel to analyze and fix detected errors. Each designer was able to analyze and correct only about three bugs per day. Since the design staff was limited (3 graduate students), the rate at which tests were executed was reduced. Time spent identifying and fixing errors added to the overall design schedule.

7.6 A (small) Theory of System Debugging

The significant error in the Aurora III development schedule prompted interest in developing a theory of design completion. Determining when a design is complete is a fundamental question in system development. A quantitative prediction of when a VLSI design project will be completed has not, to our knowledge, been previously presented.

The shape of the cumulative bug distribution curve was illustrated in Figure 7.7. The curve begins with rapid detection of the initial bugs, and the rate of finding bugs gradually slows over time. This family of curves can be fit by many equations, but one obvious choice would be an exponential.

The model assumes that bugs are uniformly distributed throughout the design, and occur with a uniform random probability. Bugs that affect a large portion of the chip are easy to find, and are detected and corrected easily. As time progresses, however, bugs become harder and harder to detect, as they become isolated to affecting only rare events and special cases. The increase in difficulty over time can be modeled using an exponential equation to fit the data:

$$Bugs = K_0 \times \left(1 - e^{\frac{-Days}{K_1}}\right) \tag{44}$$

The constant K_0 predicts how many bugs will be found, and K_1 is the time constant for the exponential decay.The time constant can be used to estimate when the design will reach a given level of design quality. Ninety percent of the bugs are found after 2.3 time con-

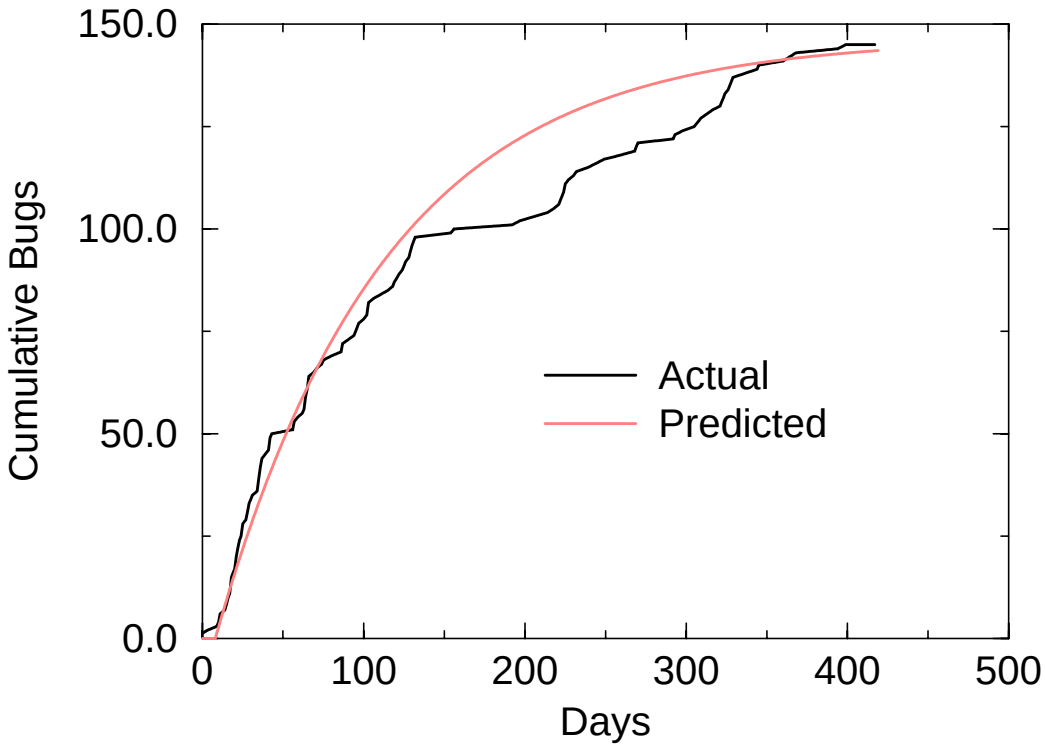

Figure 7.8 Predicted Cumulative Bug Data

stants, and 99% are found after 4.6 time constants. Fitting a curve to the cumulative bug data results in the plots shown in Figure 7.8. The curve fit is weighted to fit the peaks of the cumulative bug curve, rather than the average. The predicted line has a maximum value of 146 bugs, which is the K_0 value for the best-fit line in Equation (44). Since 145 bugs had been detected after the first 420 days of testing, few bugs remain to be found. The time constant equals 93.1 days, indicating 99% of the design errors will be found in 430 days.

However, fitting a curve after the fact does not provide any information about the project while it is underway. Also, using prediction curves generated from only the first part of the bug curve to predict the final value of the bug count can be problematic. Figure 7.9

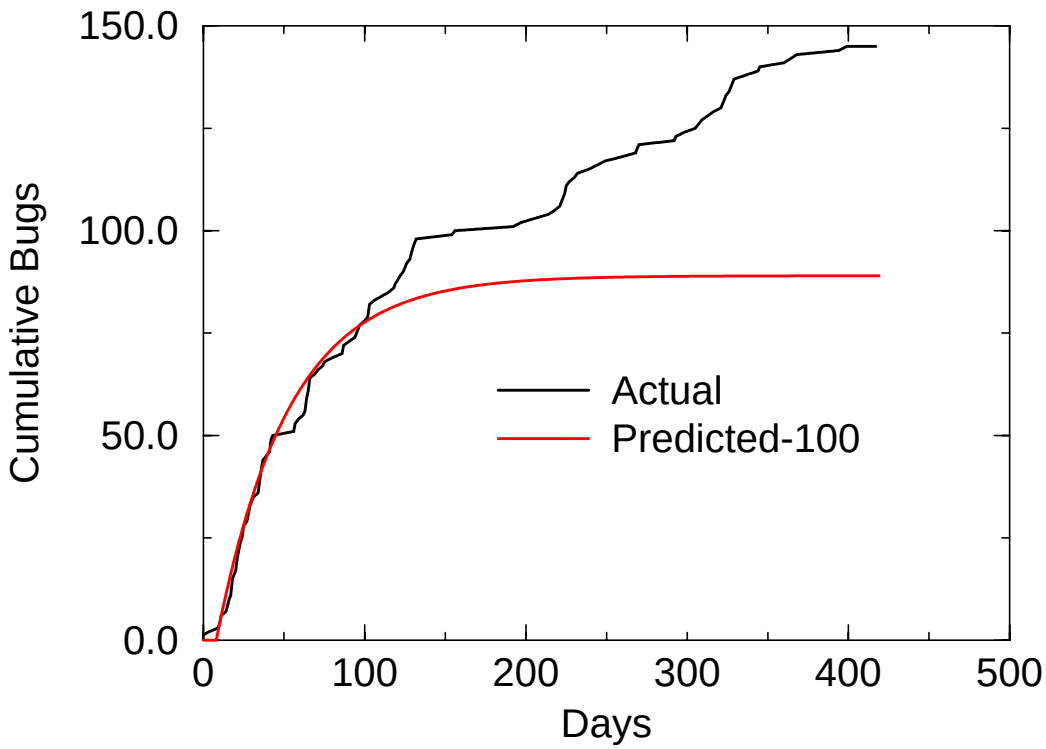

Figure 7.9 Predicted Bugs Using Only the First 100 Days of Data

shows the resulting fit if only the first 100 days of data are used to generate the fit. The fit for the known part of the graph remains good, but the bug total is dramatically underestimated.

A running prediction of the completion date can be computed by fitting a predicted bug curve to progressively larger periods. When this is graphed over the duration of the project, a progressively more accurate completion date prediction results. Figure 7.10 shows the predicted completion time graph for the Aurora III project. The X axis shows the number of days of project development effort. The Y axis is the predicted completion date for each day on the X axis. The diagonal line shows the current date. The predictions start at one quarter of the project duration, or 100 days for the Aurora III project.

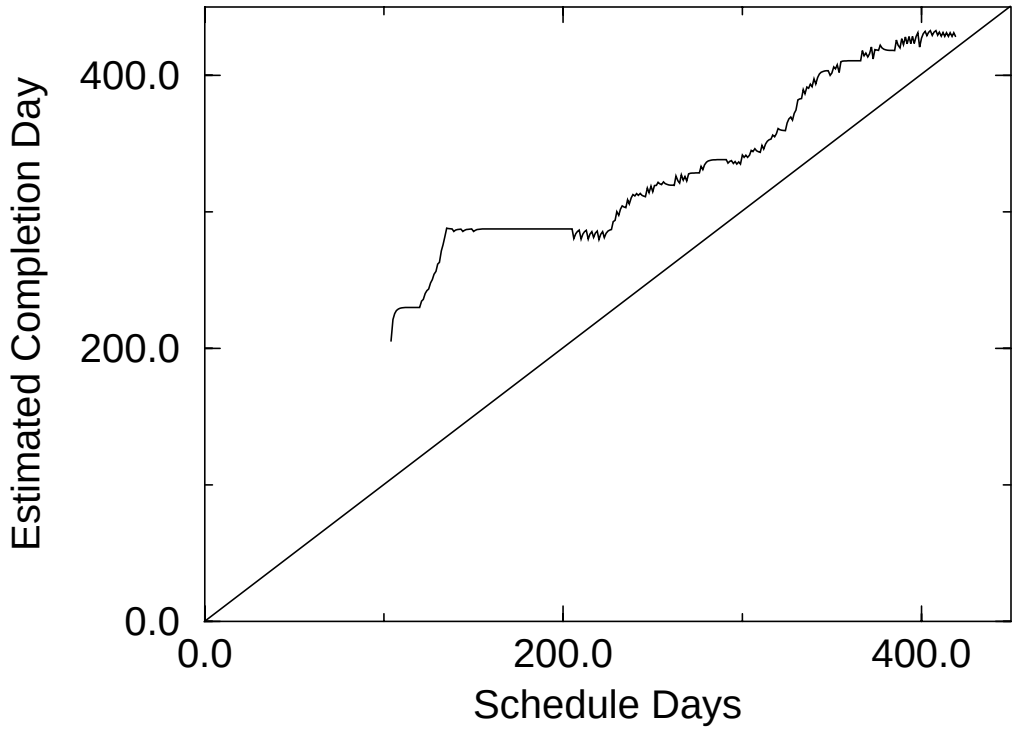

Figure 7.10 Projected Completion Date vs. Time

Initial completion estimates are optimistic. The first estimate, at 100 days, predicts a completion at day 205, as compared to 105 days from the schedule developed in Figure 7.6. As additional data are added to the graph, the estimate is rapidly updated to a more accurate estimate of 287 days. Being able to predict project completion dates is a valuable capability; only 4 months into the project, it is evident that the initial schedule is too optimistic. The running completion date can be calculated by drawing an intercept between the predicted completion date curve and the diagonal line current-time graph. Table 7.4 shows the

Day	Manual Estimate	Calculated Estimate
115	146	230
208	280	285
300	375	380
390	450	420

Table 7.4 Comparison of Estimated Completion Date

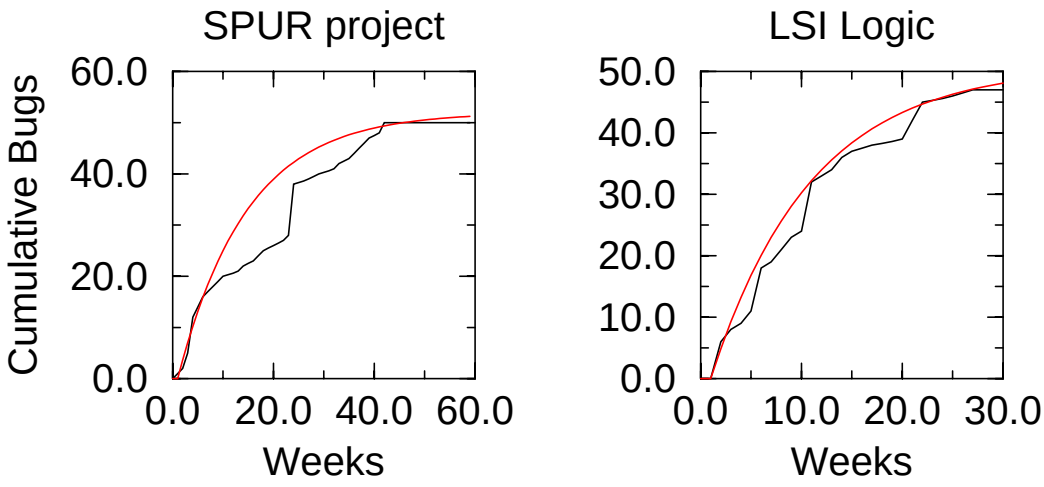

Figure 7.11 Cumulative Bug Data for Two Projects

difference between the manually generated completion date estimates and the calculated estimates. The Aurora III schedule was greatly optimistic at the beginning of the schedule, and slightly conservative at the end.

This model would be of little use if it works only for data from this project. Design process data, especially data on bugs, is closely guarded by the IC-design community. Nevertheless, two other sets of cumulative bug data were located, the first from the SPUR project at Berkeley [Wood89], and the second from a commercial microprocessor development project from LSI Logic [Peck91].

Figure 7.11 shows the cumulative bug data for the two projects. The SPUR project data has a large discontinuity at week 22 corresponding to an in-depth design review. The LSI Logic data fits the exponential prediction closely, but the chip was taped-out only two weeks after finding the last bug. A defect so close to the tapeout date results in a residual slope to the predicted bug curve. The slope of the line indicates that the model predicts 95% of the bugs have been identified, rather than the 99% level achieved by the other project curves.

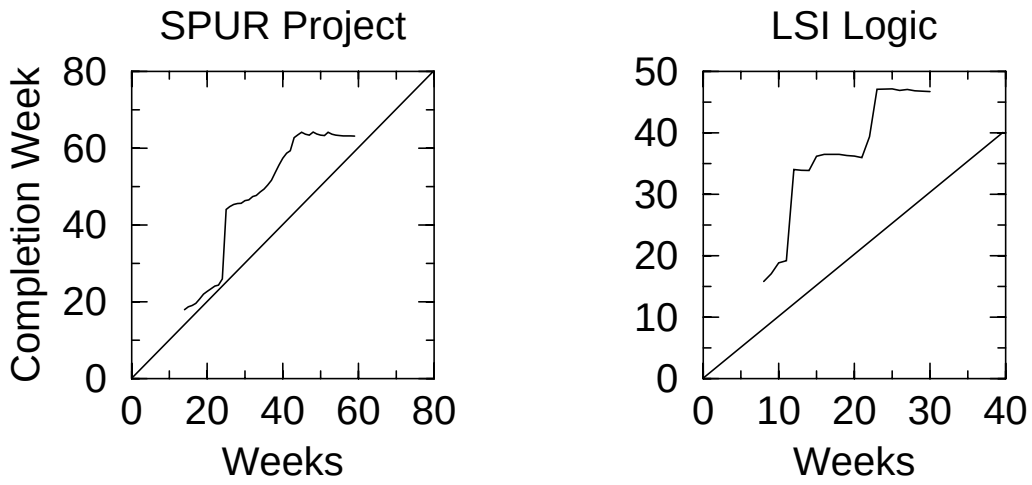

Figure 7.12 Estimated Completion Date for Two Projects

Figure 7.12 shows the estimated completion date predictions for the two projects. The estimated completion date for the SPUR project is within one week of the actual tapeout date the designers chose. The prediction for the LSI Logic project is off by nearly 50%. The error is caused by the 95% bug detection level generated by the predicted bug count curve fit.

The completion date curves for these three projects all have a similar form. The initial estimates are fairly small, and then jump to a larger target value with a significant discontinuity. The estimated date then remains stable for some time and then increases again to the final value. The final tapeout dates for these projects are all two to three times greater than the initial estimated completion dates, suggesting that a good method for projecting the completion date early in the project would be to double the initial calculated date.

7.7 Design Decision Critique

This section presents an analysis of that was good and bad in the Aurora III, with respect to both the architecture and the development process.

Throughout the design process, the designers tried not to fit too much logic into each clock cycle. In several places small optimizations were tried to cut cycles out of various paths, such as the I-cache miss fill path. Almost invariably these optimizations were later

removed because of critical-path timing constraints.

Plan to Throw One Away

Brooks advises prospective system builders to plan to discard the initial design [Brooks75]. Preliminary implementations of many of the IPU functional units provided early feedback on the implementation difficulty of the different units, and greatly helped in identifying problem areas. Early difficulties in the design of the instruction fetch unit showed this module would be the source of much of the complexity and many of the errors in the design. Although the initial design was ultimately re-designed, the initial version explored and debugged many of the issues in next address caching with integrated branch prediction.

Branch Delay Slot Difficulties

As previously mentioned, there were significantly more bugs in the IFU than were originally expected. Many of the difficulties were the result of trying to fit the architectural branch delay slot specified in the MIPS architecture into the dual issue pipeline. The delay slot adds many of the same complexities commonly associated with variable length instructions because the branch and delay slot instruction pair can span cache line, or even virtual memory pages.

Issue Model

A major research goal of this project was to determine the effect of superscalar instruction issue on clock cycle time. Initial results show that superscalar instruction issue can be incorporated into an architecture with no increase in the length of the critical paths. Superscalar issue in the Aurora III processor is simplified by a restrictive issue model.

Wiring congestion makes it difficult to provide more than one write port into the register file. Simulations showed that having only one write each cycle does not substantially increase the number of reorder-buffer-full stalls.

LSU

The LSU originally had slightly more resources allocated for non-blocking load

MSHRs than are in the final design. Trace-driven simulation showed that four MSHR entries were almost never simultaneously in use, so the number was reduced to three in the final design.

The LSU fully supports out-of-order memory requests. However the average number of outstanding requests is quite small. Because only a small number of requests are active on average, the added complexity of processing them out-of-order is probably not warranted. The PowerPC 603 design team reached this conclusion as well, and handle memory requests in order [Poursepanj94].

We originally believed that the primary benefit of non-blocking memory operations was completing ALU operations while the outstanding memory operation was processed. Because the secondary memory latency is far greater than what is supported by the depth of the reorder buffer, the operations that complete while the load is in progress make an insignificant contribution to performance. The primary benefit is the ability to generate a second cache miss while the first miss is being processed, thus processing multiple cache misses in parallel.

Function Unit Partitioning

The modular design style adopted for the IPU proved to be a highly effective way to build a large, complex chip. If the design had not been partitioned into smaller blocks, the debugging process would have been intractable. Still, the modular design style was not without drawbacks. The interfaces between modules were the source of many timing related problems, both clock phase errors and critical path lengths. A more strict enforcement of signal naming conventions would have minimized the number of phase errors.

Queue Based Communication

The queues in the processor, in the BIU and between the IPU and FPU, proved to be a valuable contribution to the design. The asynchronous nature of the BIU transmit and receive queues allowed the IPU to operate at its maximum frequency.

Prefetching

A moderate amount of prefetching is highly successful in reducing cache misses, especially for the instruction stream. In general, instruction prefetching is much more effective than data prefetching.

Pseudo-Random Testing

Pseudo-random testing proved to be a powerful method of detecting faults, especially those that resulted from interactions of multiple simultaneous events. The development process would have proceeded much more quickly if a random test structure had been developed for each module.

A random test environment was used to generate performance data for the BIU [Stanley93]. A random tester was also written for the LSU late in the design flow to facilitate regression tests during timing optimization. This tester took about 1 week to design and implement, and quickly found many errors that had not been detected by the top level random test environment.

The short time needed to write these test environments, combined with the high level of coverage they achieve at the module level, suggests these random testers should be a required element in any complex system design. Based on the bug reports for the Aurora III project, we estimate that 6 months could have been saved on the development schedule if each module had been extensively exercised in a random test environment before system integration. In retrospect this appears obvious, but day-to-day schedule pressures sometimes lead to suboptimal design and project management decisions.

7.8 A Unified CAD System for High Performance Digital IC Design

From the design experience with the Aurora III processor, a number of recommendations can be made on the structure and capabilities of an integrated CAD system for high-performance processor design. It is desirable for the layout process to be fully automatic for many reasons. Two unfortunate interactions between verification and timing optimiza-

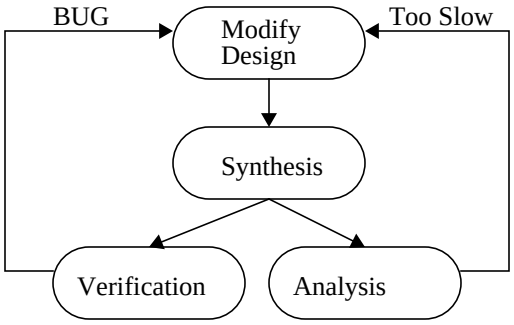

Figure 7.13 Behavioral Synthesis Design Loop

tion, illustrated in Figure 7.13, lead one to want a better integrated CAD system for the design of high performance circuits. The first case occurs when a logic design bug is found by system verification. The cause of the bug must be found, and the error corrected by redesigning some portion of the logic. Much of the performance-tuned logic in the chip is hand optimized using Karnaugh maps to minimize the number of logic levels on critical paths. If the error affects any of this hand-crafted logic, the entire section must be redesigned by hand.

The second interaction between design and verification is the complementary case. If a performance problem is corrected through modification of the Verilog code, the logic must be reverified and the design run through a series of regression tests. After each logic change, a week or more of verification was required for the designers to be confident that the modification did not subtly affect some unintended portion of the design. This problem becomes worse as the design nears completion. The final set of bugs involve combinations that are so rare as to only appear every 50 to 100 Million instructions. The ability to ensure that fixing a bug in an obscure case has not introduced a bug in some other equally obscure case is an open research problem.

Figure 7.14 shows a proposed CAD system for the design of high performance systems. Executable programs are shown as rectangles in the diagram. Shaded rectangles have been prototyped in the current CAD system.

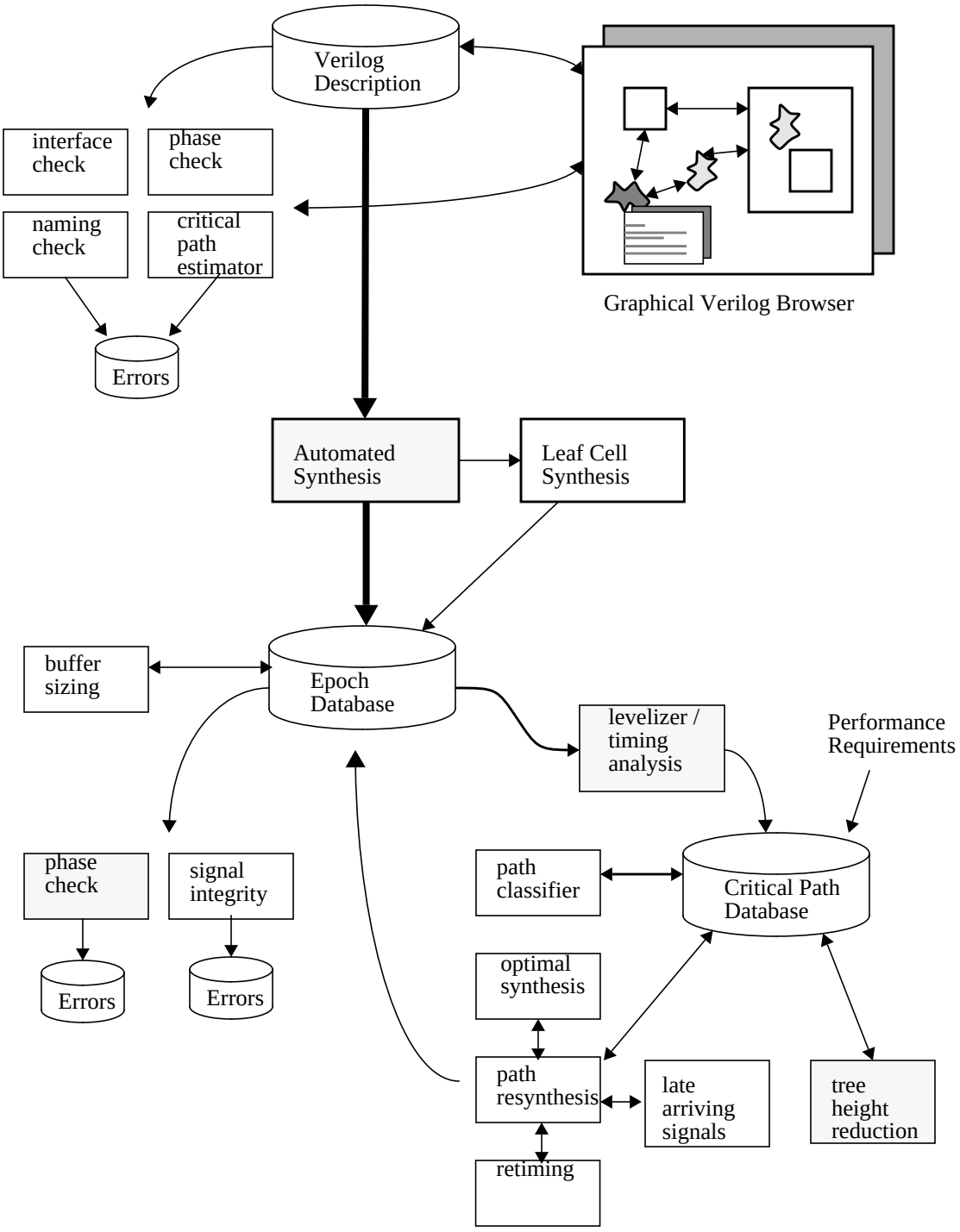

Figure 7.14 Proposed CAD System

7.8.1 Verilog Analysis Tools

The proposed design system has three main components: a Verilog description, a gate level database, and a database of timing information and optimization tools. Most user interaction is with a high level description of the processor in the Verilog language.

Verilog Description: The primary interaction with the architectural description is through the Verilog hardware description language. Several utilities and tools would help reduce the number of design errors and facilitate rapid verification and modification of this model. The initial Verilog model could be translated from the machine description files used for trace driven simulation.

Interface Checking: To maintain a high clock rate, it is important to monitor the interface between different processor modules. It is easy to inadvertently add delay to a signal that leaves a module without realizing the effect this will have on other modules in the design. This tool would scan the Verilog description to ensure that all signals entering and leaving a module meet proper design guidelines. An example guideline would require all signals leaving a module to be latched.

Signal Name Checking: A signal naming convention should be adopted early in the design, and enforced using tools to monitor whether individual components adhere to the naming standards [Karplus84]. Failure to maintain consistent naming results in errors and confusion later.

Phase Checking: Significant time can be saved in the design process if the design can be created as correctly as possible. Phase errors were one of the more frequently occurring errors in the Aurora III design. These errors were checked at the gate level in the Aurora III design, but many phase related checks could be performed at the Verilog description level, reducing the design iteration time.

Critical Path Estimator: Significant hand optimization was required on the Aurora III design to reduce the length of critical paths. Most of the paths could have been easily detected early in the design using only high level Verilog descriptions. This

tool would look at the Verilog and estimate the length of logic paths, assign delays for structural components using tables, and estimate the complexity of behavioral block with heuristics.

The high level Verilog description is translated using existing commercial tools into a gate level netlist.

7.8.2 Layout Synthesis

The layouts for current microprocessor designs are composed of between one hundred and many hundred library cells. Although the design may contain hundreds of different layout cell types, the majority of the design is often accounted for with just a few cells. A cell that is used infrequently requires nearly as much design effort as a cell that is used thousands of times in a design. As a result, the majority of the layout design effort is devoted to a minority of the resulting chip area, simply because a few frequently occurring cells occupy most of the layout area.

A solution to this problem is to automatically generate the cells that account for only a small percentage of the final chip area. Although these cells would not be as dense as manually created cells, the time saved by reducing the size of the layout library could be devoted to optimizing the cells that occur more frequently.

7.8.3 Gate Level Optimization

Gate level optimization is time consuming and error prone. Some of the more important design features requiring automated checking and optimization include:

Phase Checking: Phase checks can be performed at both the behavioral and the gate levels. Checking at the gate level can be more thorough, and can include support for more complicated constructs, such as gated clocks.

Signal Integrity Checking: Continued reduction in IC process metallization dimensions is increasing the resistance and capacitance of interconnect routing, causing problems in clock skew, reliability, and noise coupling between adjacent

signals. Several tools are needed that would identify and correct these problems after layout.

Levelizer/Timing Analysis: Determining where to devote effort in the design to achieve the largest performance payoff is a challenge. Performance of the processor is determined by the length of the critical paths and the parasitic loading of these paths. Critical path optimization can be performed at two levels, a high level in which the parasitics are ignored and typical values for wire length and output drive are assumed, and a detailed level in which the drive capabilities of the individual gates are considered, along with the detailed routing of the interconnect nets. The high-level estimate is useful during the early stages of design optimization, when many iterations of performance tuning are required. The detailed analysis can take many hours to perform, but provides a more complete picture of the timing behavior of the chip.

Path Classifier: After the timing tools have identified the critical components in the design, the path classifier decides what corrective action to take for each path. Several possible solutions exist, depending on the configuration of the logic. Example optimizations include resynthesis, tree height reduction, late arriving signal optimization and optimal synthesis.

Path Resynthesis: The modular nature of the design results in paths between multiple synthesized blocks. Some of these logic paths can be shortened by extracting the long path from the rest of the logic and resynthesizing the path as a single block of logic.

Tree Height Reduction: For many reasons, the logic in a complicated processor is not optimal. Because of the signal conventions adopted there may be redundant inverters between blocks of logic. A reduction in logic depth is often achieved by selectively increasing the fan-in of some gates. It is not advisable to do this everywhere because the net loading increases for the affected nets.

Late Arriving Signal Optimization: Not all signals arrive at a block of logic si-

multaneously. Some signals, such as cache miss signals, arrive late in the clock phase. Logic affected by these signals can be restructured so the minimum amount of computation is required after the late signal arrives.

Optimal Synthesis: Many of the small logic blocks in critical paths were hand designed using manual logic minimization methods. These blocks were usually simple, having fewer than 7 inputs and two or three outputs. These block are good candidates for optimal synthesis, using either exhaustive or branch and bound methods [Davidson69].

Retiming Optimization: A long logic path is often proceeded by a much shorter path in the pervious clock phase. The long path length can be reduced by moving the early part of the long path before the latch dividing the short from the long path [Lockyear93].

Many of the tools proposed here automate tasks that were performed by hand in the Aurora III design. However, prototypes of some of the tools were constructed, and were vital in producing the final version of the chip.

7.9 Summary

This chapter described the design process for the Aurora III project. A verification methodology based on self-testing random programs was presented. The difficulty in creating an accurate schedule was described, motivating a theory of design completion. The theory of design completion was used to model the Aurora III and two other VLSI design projects, providing good estimates of completion dates early in the project. A critique of the Aurora III architecture was presented, and a unified CAD system was proposed.

CHAPTER 8
Conclusion

8.1 Future Work

The micro-architecture developed for this dissertation can be extended to make it both faster and more flexible. Though the current IPU makes extensive use of queues to support multiple memory fetches, even more decoupling is possible. Figure 8.1 shows two locations where queues could be added to the existing architecture. The diagram on the left of the figure shows transmit and receive queues in the bus interface unit and the MSHR entries in the load/store unit. The diagram on the right shows additional queues between the instruction fetch unit and execute units, and between the execute unit and the load/store unit.

The current Aurora III design stalls instruction fetch whenever a load interlock occurs, often delaying the detection of an instruction cache miss. The queues between the IFU and the IEU provide a buffer to hold instructions while the IFU fetches ahead to the next cache line. If an instruction miss occurs, the miss will occur earlier, and will possibly overlap with a concurrent data cache fetch.

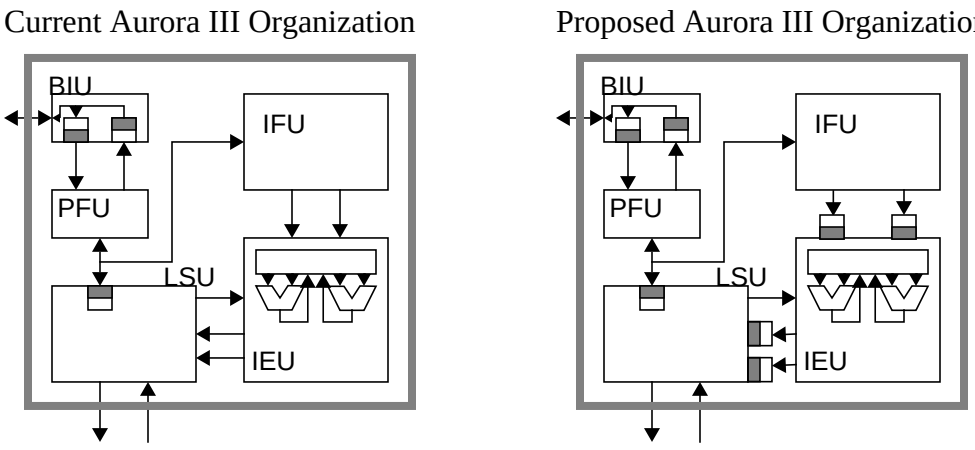

Figure 8.1 Proposed Additional Queue Locations

One of the main issue constraints in the current Aurora III design is that only one memory instruction may be issued each cycle. Most memory accesses are loads, and the pipeline provides sufficient adders and busses to process two loads each cycle, but the LSU has only a single access port. Adding the queue before the LSU would allow multiple loads to be executed in the IEU each cycle.

Architectural Optimization

The resource allocation studies relied on manual selection of the candidate machines. The configurations could be selected automatically using an optimization algorithm to guide the selection. Stanley has reported some initial results of automated micro-architecture optimization [Stanley95].

Random Testing

An interesting extension of the random testing work is in the area of automated system level design verification. The problem of architectural design verification, ensuring that a design conforms to a specification, is largely unsolved.

DeMillo has proposed a theory of algorithm verification involving the random permutation of source code [DeMillo78, Maurer88]. His results show that algorithms that contain errors usually only differ from correct systems by a few statements. Random modifications of the algorithm source code are applied, and a set of test vectors are run on the modified algorithm. If new errors are generated, it shows the modified source lines are necessary for the correct operation of the algorithm. If the modified line does not change the functionality of the program, the line may be redundant, and might be removed.

8.2 Research Contributions

This work has made original research contributions in three major areas: processor microarchitecture, CAD tools, and circuit design.

Microarchitecture contributions

1. **Evaluated GaAs DCFL for Microprocessor Design:** Gallium Arsenide E/D

MESFET logic was evaluated as a technology for constructing pipelined microprocessors. A model for process density was developed and used to identify critical process sensitivities. Using these sensitivities, a microarchitecture of a prototype microprocessor was optimized to give high performance. Several weaknesses were identified in the technology, from both economic and technical perspectives.

Economically, GaAs has a much smaller investment base than silicon, reducing the total process research and development funds invested in new processes development. This causes GaAs processes to lag silicon in interconnect metallization technology, reducing the performance benefit of faster transistors. The material has higher defect levels than silicon and the wafers are much more expensive, resulting in significantly higher component prices for devices as complicated as microprocessors.

From a technical perspective, DCFL has a number of shortcomings compared to CMOS. The restriction of NOR-gate-only logic increases the gate count and results in longer critical paths than in CMOS. Low noise margins and wide process variations make it difficult to build fast memories, a fundamental component for high speed processors.

Table 8.1 reexamines the microprocessor design support offered by the three technologies listed in Table 1.1, plus GaAs DCFL. When the technology attributes needed to build a competitive microprocessor are examined, it is found that GaAs

Technology Attribute	CMOS	NMOS	ECL	GaAs DCFL
Gate speed	Good	Fair	Excellent	Excellent
Circuit density	Good	Excellent	Fair	Fair
On-chip Memory	Good	Fair	Fair	Poor
Drive capability	Good	Good	Excellent	Fair
I/O bandwidth	Good	Fair	Excellent	Excellent
Power Dissipation	Good	Fair	Poor	Poor
Clocking	Good	Fair	Good	Poor

Table 8.1 Technology Support for Microprocessor Development

DCFL rates only fair. Because of the difficulties in building GaAs circuits, and the fact that the systems produced are no faster than top end CMOS systems, it is unlikely that GaAs DCFL will find a place in the microprocessor design community.

2. **Verified Fast Superscalar Issue:** Despite the challenges of GaAs for building processors, a highly tuned superscalar microprocessor has been designed and evaluated. The processor does not suffer any critical path length penalty for supporting superscalar instruction issue.

   A decoded instruction cache with next-address caching speeds instruction issue by precomputing the branch target address. A restrictive issue model is a vital component in achieving low-overhead instruction level parallelism.

   A separate Load/Store Unit isolates the complexity of memory operations to a separate function unit, allowing a shorter integer execution pipeline with reduced parasitic loading.

3. **Developed Next Address Caching:** As part of this research, a method of eliminating the branch delay slot for pipelined multi-issue processors was developed. This scheme, called Next Address Caching, was concurrently developed by engineers at SGI and Sun [Hsu94, Agrawal94]. Next address caching places the predicted target address in the tag field of the instruction cache; branch instructions use this predicted value to address the cache without waiting for the branch addition to calculate the target address.

4. **Invented Direct Mapped Reorder Buffer:** To avoid associative lookup, a function that GaAs performs poorly, a direct mapped register renaming scheme was developed.

5. **Evaluated Impact of Continued Processor Clock Frequency Growth:** As described in Chaper 2, processor clock frequency growth has forced the adoption of more sophisticated memory systems, including the support for multiple outstanding cache misses and hardware prefetching.

   The Aurora III pipeline is not able to use deeper reorder buffers or more non-

blocking capability because the load-use interlocks in the pipeline cause instruction issue to stall before the additional capability would be needed. Continued performance growth will require additional design changes, such as out-of-order memory requests.

There is an inherent conflict between wide issue and fast cycle time. Extracting the optimum parallelism to maximize system performance remains an interesting research topic. As shown in Chapter 6, longer cache access latencies do impact the amount of parallelism that can be efficiently exploited.

CAD Tool Contributions

6. **Optimal Datapath Placement:** The placement algorithms developed for reducing the area and netlengths of datapath modules in this project have been adopted by industry. These algorithms result in significant improvements in all measures of layout design quality, including area, netlength, power and yield.

7. **Developed Theory of Design Completion:** A theory of system debugging has been developed and shows good agreement with data from this and two previous projects. A quantitative method of predicting the completion date for a VLSI design project has been presented.

Circuit Contributions

8. **Developed GaAs Circuit Elements:** Many logic elements for high performance computers have been developed. Most of the basic building blocks have been highly tuned to produce dense, fast layouts. Example elements include a pipelined Ling adder, a multiplexor-based barrel shifter, and a number of Earle latches incorporating logic as well as latching capability. A DCFL tristate buffer was developed to reduce routing congestion and allow many sources to drive a destination register. This circuit met the goals of simultaneously reducing both the routing congestion and the logic delay.

8.3 Outlooks for Microprocessors

As shown in Chapter 1, microprocessors have undergone significant performance improvements in recent years. These improvements show no signs of diminishing, although the economic investment needed to sustain the current rate of growth is large. Technology limits show no signs of being reached in the next few years. The outlook for continued CMOS processor performance improvements is exellent. These improvements will come from a combination of technology and architectural advances. Processor clock rates will continue to increase dramatically, with growth rates remaining near the 40% per year historical trends shown in Figure 1.3.

8.4 GaAs Microprocessors and Market Entry Dynamics

Throughout its 25 year history as a semiconductor technology, GaAs has been touted as the “Technology of the Future”. Although it seems to have many performance advantages over silicon, market success for digital circuits has remained elusive, though GaAs does have the majority of the opto-electronic and microwave markets. What factors have contributed to the slow acceptance of GaAs by the digital marketplace?

Powerful forces come into play when a new technology with many apparent or advertised advantages enters the marketplace. Sites has proposed a model for market dynamics that is composed of four phases: Laboratory, Market Entry, Response, and Resolution [Sites93]. Sites looked at market share as a function of time for several new technology introductions. Figure 8.2 shows an example market entry graph for the introduction of CMOS technology into the high-performance circuit market previously dominated by ECL. The graph shows market share vs. time for the competing technologies. The graphs are of schematic value only; the precise X values and crossing dates are not known. The four phases of market dynamics are shown in Figure 8.3. At the left side of the graph, the new technology is in an advanced stage of laboratory development. In this phase the older technology has nearly all of the market, and the new technology exists only in special-purpose, niche hardware, often as a technology demonstration vehicle.

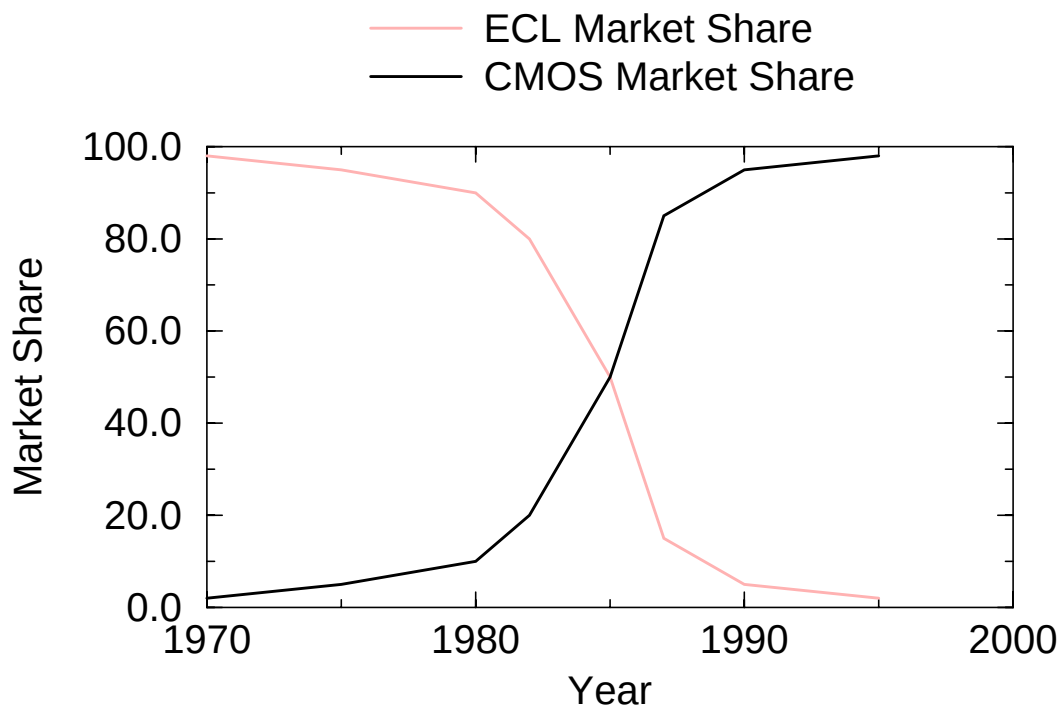

Figure 8.2 ECL vs. CMOS Market Response Graph

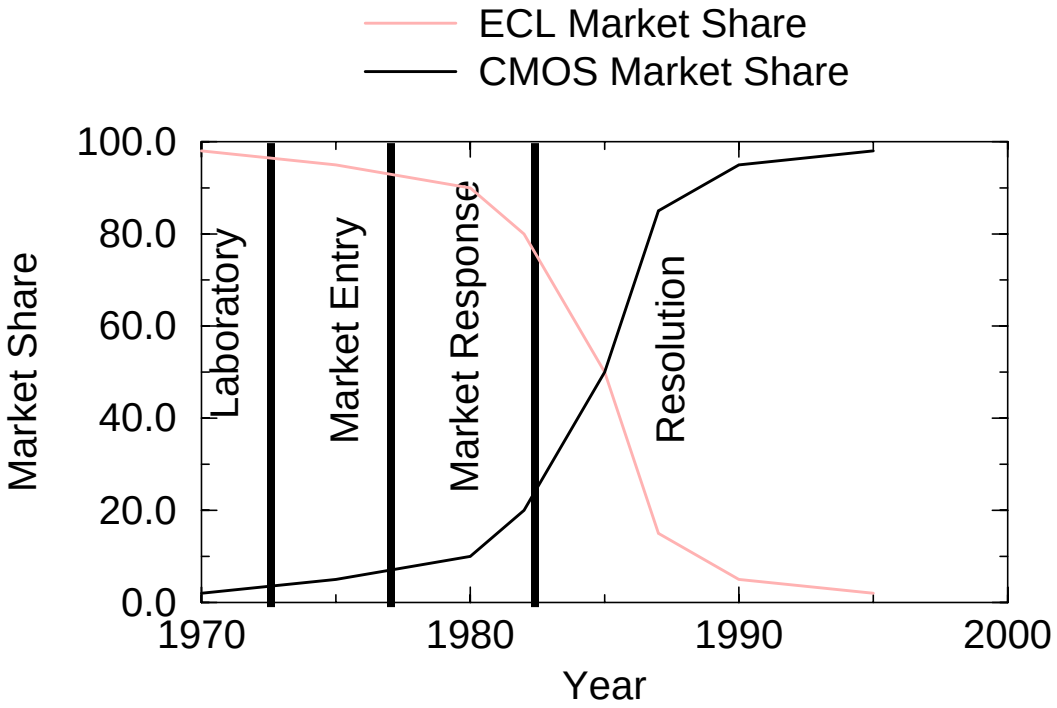

Figure 8.3 Four Phases of Market Dynamics

If the new technology shows promise, it will enter the marketplace and will begin to compete with established technologies for market share. This phase is known as market entry. Early in this phase, the new technology can achieve large percentage gains in market share, although the actual market share remains small.

As the new technology begins to gain market share, the established technologies will be forced to react by either improving performance or reducing price. This reaction is known as market response. An example of market response can be found in the cost of core memory modules during the introduction of the MOS Dynamic RAM by Intel in the early 1970's. Intel found that when they went to market with the 1103 4K-bit dynamic RAM, core memory vendors had reduced their prices to 1/3 of their previous cost, making design wins for the DRAM much harder.

The final phase of market dynamics is resolution. There are three possible outcomes to the market share competition. The existing technology may improve its performance and cost dramatically and drive the new technology out of the market. The new technology may have fundamental benefits that cannot be offered by the existing technology and will replace the old technology. Or, the market may be split, with both technologies offering advantages to different market segments, in which case neither technology will achieve a dominant position.

8.4.1 Market Response Example: Exotic Photolithography

An example of the aggressive response of an existing technology to retain market share is found in the IC photolithography field. It has been clear for some time that the minimum feature size of ICs will continue to decrease. Figure 8.4 shows the feature size trend.In the mid 1980's there was much concern about how lines smaller than about 0.7 micron could be produced. The wavelength of light used to pattern IC's is about 0.38 microns. Physical laws say that a pattern smaller than 2 wavelengths of light cannot be resolved optically. There was concern that inability to pattern small features would slow the continued decrease in process feature size, requiring an expensive solution such as direct-write electron

beam or X-ray lithography.

A switch to a new patterning technology would render much of the current $100 billion installed base of equipment obsolete. This was a powerful incentive to develop a solution that would allow existing equipment to be used.

The solution developed uses phase shifters etched from the glass masks to create optical interference patterns in exactly the proper places on the IC. This technique, called Optical Phase Shift Lithography, allows patterning to slightly less than one wavelength of light. Research is also continuing on sources and optics for shorter wavelengths. Using these techniques, the crossover point for exotic lithographs has been moved out to at least the 0.15 micron generation, extending the lifetime of optical techniques by nearly 8 years.

The adoption of more exotic techniques has not been eliminated. The underlaying physical limits have not been removed, but the enormous economic incentive to find a solution prompted research into ways to use existing technology. The adoption of more advanced lithographic techniques has been delayed by dramatic improvements in existing technology.

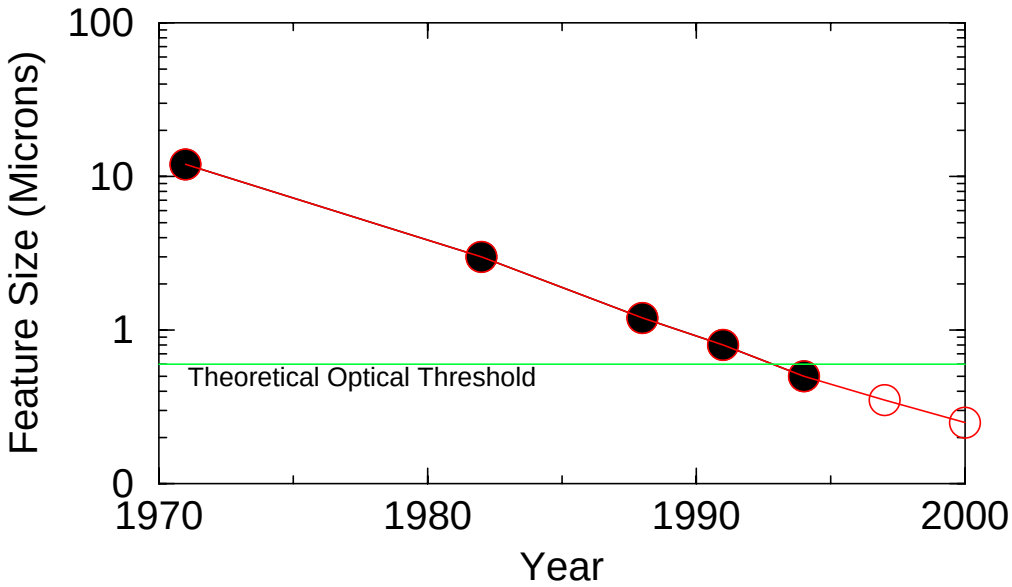

Figure 8.4 Minimum Feature Size

8.4.2 Market Capture Example: NMOS vs. CMOS 1983

An example of a new technology capturing a market can be found in the introduction of CMOS technology in the early 1980's. At that time, three main integrated circuit fabrication technologies were in use: NMOS, CMOS and bipolar. Bipolar technology was used primarily in the fabrication of LSI standard parts such as NAND gates and individual flip-flops, and was also used extensively in mainframe computers. A furious battle raged over which technology was better for the design of VLSI components like microprocessors. NMOS had the largest share of the market, being the primary technology for microprocessor design in the 1970's. NMOS seemed to have many advantages over CMOS, including fewer processing steps, about half the transistor count for similar functions, and higher density. CMOS had one primary advantage: lower power dissipation.

Technical publications often carried articles detailing the reasons why CMOS would never be commercially viable. The continued increase in IC complexity predicted by Moore's law ultimately decided which technology was better. Microprocessor chips became so large that the power requirement became more important than the area of the chip. The area of complex circuits is usually set by the interconnect routing rather than by the transistors, reducing the penalty for the larger transistor count. The lower power dissipation of CMOS allowed the design of larger chips with more functionality.

Requirements for lower power dissipation are now causing companies to drop BiCMOS technology for microprocessors, in some cases after substantial recent investment. The current generation of Intel Pentium is built on a BiCMOS process, and it is believed that the next major processor, the P6, will be BiCMOS as well [Colwell95]. However, it appears that after this next generation, all major microprocessors will be pure CMOS designs. The cause of this switch is the drop in power supply voltage needed to fit 10 million transistors on a single chip with manageable power levels, which has eliminated the performance advantage of the bipolar gates. Combined with lower yields, this makes BiCMOS technology less attractive. Other companies, such as Texas Instruments, have also an-

nounced a move back to CMOS only logic.

The move from BiCMOS back to CMOS follows the move from Bipolar logic to CMOS, both in MSI level components and in mainframes and supercomputers. The change from Bipolar to CMOS MSI logic has been driven by a combination of cost and power consumption considerations. The change to CMOS logic in large scale computing, such as IBM's new ES9000, is more surprising, since the goal of these machines is high performance without concern for power dissipation or cost. Even without power or cost constraints, CMOS has become the preferred technology.

In 1993, total worldwide capital expenditures on CMOS technology development exceeded 18 billion dollars. Intel alone spent $1.9 billion on new plants and equipment. This level of investment ensures innovative solutions to the technical problems created by ever advancing technology.

8.5 Whither GaAs?

GaAs technology faces many challenges from an economic standpoint. The total GaAs semiconductor market is between 100 and 200 million dollars. This is less than 1% of the world semiconductor market. The enormous investment in CMOS technologies accelerates its advances, while the level of investment for GaAs places it in a permanent catch-up position. This effect has recently been studied as the economics of increasing returns, or positive feedback economics [Arthur90].

There are three fundamental ways that CMOS domination of the electronics marketplace hurts GaAs. First, as mentioned earlier, GaAs is a brittle physical material. The maximum wafer size is limited by the strength of the material, and is currently 150mm in diameter. Most GaAs fabrication currently uses 100mm wafers. Current silicon wafers are either 150mm or 200mm in diameter, and this size continues to increase. Fabrication equipment is optimized for the larger silicon market. New equipment is developed for the standard wafer sizes. If GaAs cannot keep up in wafer size, manufacturing must be done using older equipment, severely impacting the quality of the results.

Secondly, because silicon uses larger wafers, the fabrication efficiency is greater, lowering the manufacturing cost. GaAs requires fewer fabrication steps than silicon, but is again hampered by its much smaller market, preventing it from achieving the same economies of scale as silicon.

Finally, existing GaAs technologies dissipate too much power. Fast circuits are possible, but the power dissipation makes the resulting solutions unattractive except in niche markets such as supercomputers.

To compete with CMOS processors, GaAs must provide a compelling advantage in either performance, price, power or reliability. GaAs DCFL provides none of these charateristics. Newer technologies, such as Complementary GaAs may fare better. New applications, such as portable graphic oriented computers and personal digital assistants (PDAs), offer one possible market where high performance and low power are required. If there is hope for Gallium Arsenide as a processor technology, it must be in this high performance, low power area of applications.