JULY 1989

WRL
Research Report 89/11

A 20-MIPS Sustained 32-bit CMOS Microprocessor with High Ratio of Sustained to Peak Performance

Norman P. Jouppi and Jeffrey Y. F. Tang

digital **Western Research Laboratory** 100 Hamilton Avenue Palo Alto, California 94301 USA

A 20-MIPS Sustained 32-bit CMOS Microprocessor with High Ratio of Sustained to Peak Performance

Norman P. Jouppi and Jeffrey Y. F. Tang

July, 1989

Abstract

This paper describes the design of a CPU chip with a high ratio of sustained system to peak performance (0.80). Attaining a high ratio of sustained system performance to peak performance avoids wasting circuit design effort at an architectural level through inefficient use of machine resources. The chip contains 180K transistors on a 6.98mm X 8.73mm die utilizing a 1.5um CMOS process to obtain a sustained system performance of 20 MIPS. By keeping the design simple and regular both at an architectural and circuit level, and by using high-level tools on the complete design, a high sustained performance was obtained with relatively little design effort (2.5 man years).

This is a preprint of a paper that will appear in the *IEEE Journal of Solid State Circuits*, Special Issue on Logic and Memory, October, 1989.

1. Introduction

Peak performance of processors is widely quoted. This is unfortunate because peak performance is often much higher than the sustained performance obtained in a system environment executing large benchmarks. Peak performance is used whenever different architectures are compared based on their clock speed, or processors are compared based on peak MIPS, MFLOPS, or MOPS. The peak performance of several recent microprocessors is five times that of their sustained system performance [9, 12], and factors of two or more are common.

Although the sustained performance is not easily related to a machines's peak performance, the cost of a design often is. This is because the components of a machine must be designed to support the peak performance. For example, if a machine has a peak execution rate of 50 million instructions per second, its instruction cache must be designed to provide 50 million accesses per second. Similarly, busses that cross chip boundaries also usually must operate a frequency given by the peak performance of the machine.

If the ratio of sustained performance to peak performance is less than one (and it must be at least a little less than one for all practical machines), then some of the resources of the machine are effectively being wasted. For example, a machine that can issue up to three instructions per cycle that averages 1.0 cycles per instruction (CPI) has a ratio of sustained to peak performance of 1/3. These wasted resources can be very costly, since the cost of resources is non-linear with their performance at the high end. For example, a super-fast cache that operates twice as fast as an already fast cache is probably more than twice as expensive as the fast cache, if it can be implemented at all. In terms of on-chip circuits, a sense amp that is twice is fast as an already fast design is likely to be much more complex and have much reduced margins. Not only are faster components more expensive, but the design time required to design circuits increases as they become faster. For these reasons a given sustained performance should be obtained with the lowest possible peak performance.

One measure of peak to sustained performance for machines that issue at most one instruction per cycle is the average number of cycles per instruction (CPI). Typical CPI figures for CISC machines are quite high; for example the last three single-chip VAX computers have all had CPI's greater than 10 [1, 11, 2]. This means that resources used once per instruction are wasted 90% of the time! In this light, one of the big advances of RISC machines is to increase the utilization of the underlying hardware by reducing the CPI. Many RISC machines have CPI's from 2 to 5 [12], while some approach 1.5 [10]. For machines that can issue more than one instruction per cycle, the CPI must be multiplied by the maximum number of instructions that can be issued per cycle to obtain the ratio of peak to sustained performance.

This paper describes a CPU chip and system, called the MultiTitan, designed to obtain a high sustained performance concurrent with high utilization of machine resources by having a high ratio of sustained to peak performance. In a system environment running large benchmarks and counting all external and internal cache misses, interlocks, and stalls, we were able to obtain a sustained to peak performance ratio of 0.80 (i.e., 1.25 CPI). This corresponds to an effective machine utilization of 80% for resources that are used once per instruction.

A second way in which machine resources can be wasted is by having specialized circuitry which is only used for certain instructions. For example, a machine might provide different adders for branch displacements, load displacements, and general ALU operations. But since branches, loads, and ALU operations each require only one addition per instruction, providing specialized adders decreases the utilization of each and increases the machine resources required. "Special cases," whether in data path or control, add extra hardware to the machine which is infrequently utilized. Thus, in the design of the MultiTitan CPU we also tried to build a simple regular machine with very few special cases in order to further increase the efficiency of the machine.

Finally, designers obtain more leverage for their effort in machines which are simple and regular. This is because improving the circuit performance of one part of a simple and regular machine is likely to improve the performance of most instructions. Improving the performance of one part of machine with many special cases is likely to improve only the special cases for which the part is provided.

2. Architecture

Architecturally each processor of the MultiTitan is similar in some respects to the DECWRL Titan [7], which was built from 100K ECL MSI parts. Like the ECL Titan, the MultiTitan is a very simple RISC machine with a branch delay of one. However, the MultiTitan is not object-code-compatible with the ECL Titan. Unlike the ECL Titan, the MultiTitan has hardware support for fine-grain parallel processing, unified vector/scalar floating point registers and operations, and a different pipeline and method for handling exceptions.

2.1. System overview

Figure 1 is an overview of one MultiTitan processor. Each processor consists of three custom chips: the CPU, floating point coprocessor (FPU), and external cache controller (CCU).

The external system data bus is 64b wide. This allows full-precision floating-point operands to be accessed in one cycle by the floating-point coprocessor. It also allows the instruction cache to be refilled at a rate of two instructions per cycle. Thus the four-instruction cache line is refilled in two cycles. The 64b bus significantly improves the performance of large benchmarks and floating-point intensive benchmarks. For example, the double-precision vector Linpack floating-point performance is improved by over 50% to 6 MFLOPS by the use of a 64b data bus. Similarly, the reduction in cache miss refill time from 4 cycles to 2 cycles reduces the instruction cache miss contribution from 0.064 CPI (cycles per instruction) to 0.032, improving the sustained performance approximately 3% for a given peak performance.

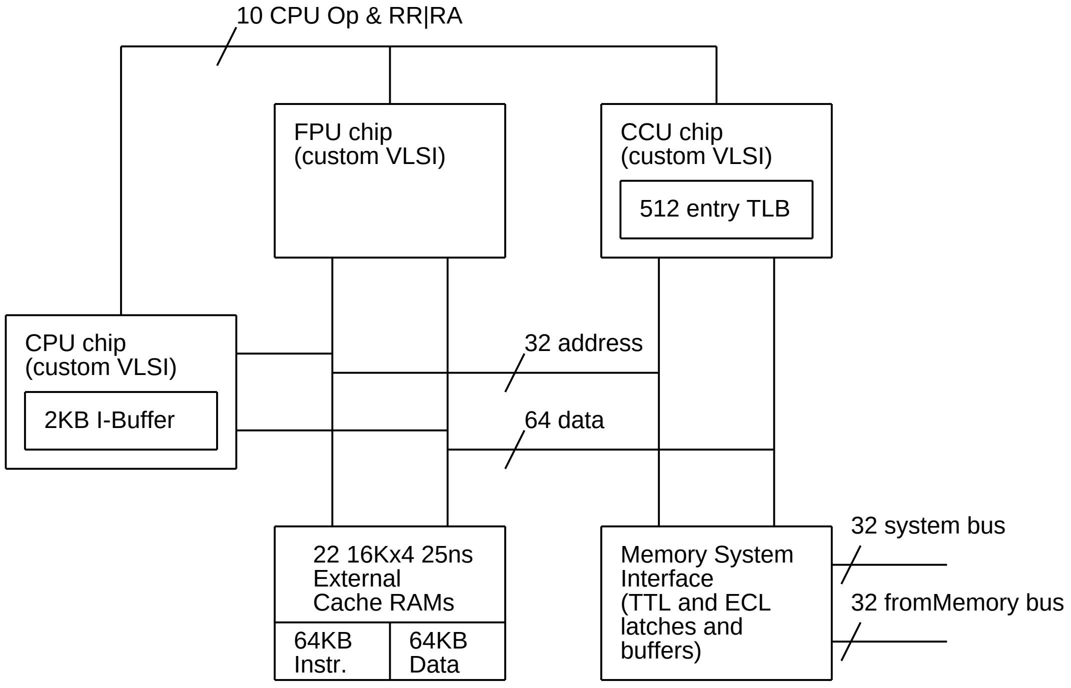


Figure 1: One MultiTitan processor

2.2. Instruction set

The MultiTitan instruction set consists of two instruction formats (see Figure 2). Registers to be fetched in the instruction fetch stage appear in the same place in both formats, so decoding of the format is not required before the access of the register file begins. For example, the register file always fetches the registers specified by the bits in the ra and rb positions, even if the instruction is in the immediate format. Similarly rr is always accessed in the machine's last pipestage, whether written for ALU operations and loads or read for stores.

A summary of the instruction set is presented in Table 1. The encoding of the instruction fields was chosen to simplify decoding as much as possible. Although the opcode is four bits, many control functions can be determined by examining only a subset of the opcode bits. For example, all instructions in the immediate format have the most significant bit of their opcode equal to 1, while those in the register format have 0 as their most significant bit. All control information is decoded in the instruction fetch stage. Control bits that are used in later pipestages are delayed by shift registers until they are needed. All control signals can be generated by the instruction decode using at most two levels of four-input logic and an inverter. A total of 74 gates are required in all of the instruction decode logic.

3. Implementation

The processor has been implemented in a 1.5u CMOS process. Typical process parameters are given in Table 2. Table 3 summarizes major characteristics of the chip. A photomicrograph of the chip is given in Figure 3.

The distribution of transistors within the chip is given in Table 4. 75% of the transistors are used in the instruction cache, and approximately 10% are used in the register file. The control

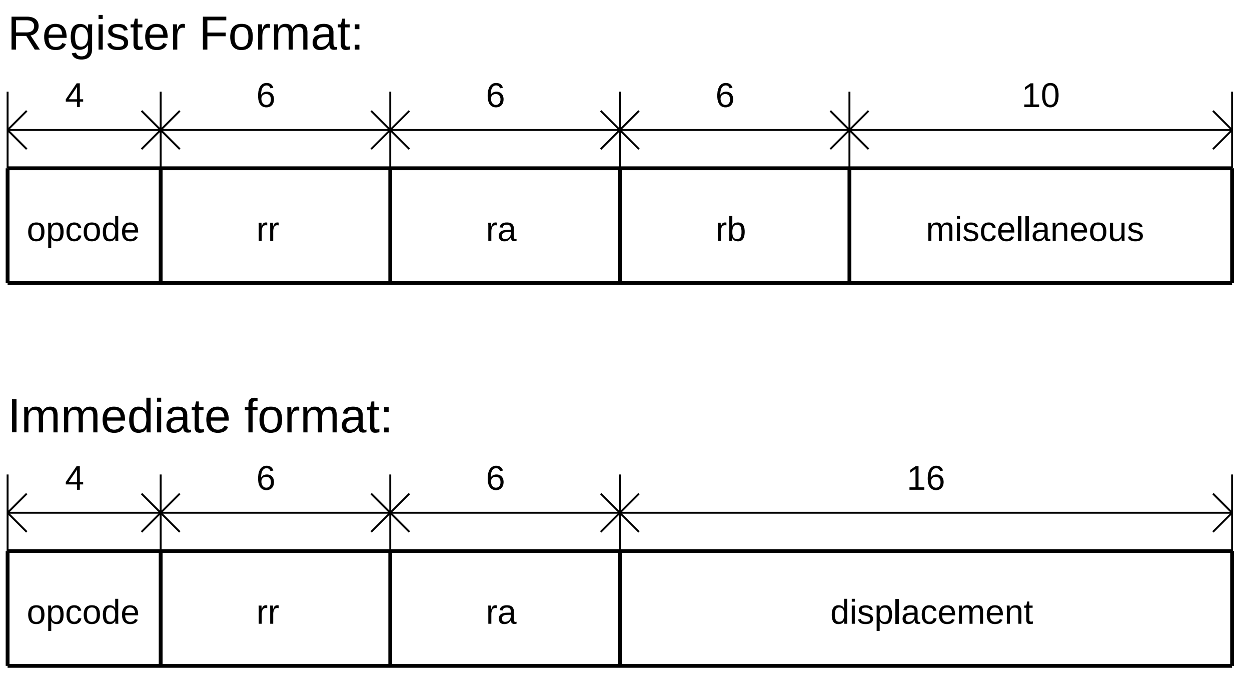


Figure 2: MultiTitan instruction formats

register format: opcode	instruction	immediate format: opcode	instruction
0	trap	8	reserved
1	extract	9	test (cop ALU inverse)
2	undefined	10	coprocessor store (64b)
3	variable extract	11	coprocessor load (64b)
4	coprocessor->CPU xfer	12	CPU store
5	CPU->coprocessor xfer	13	CPU load
6	coprocessor ALU	14	conditional jump
7	CPU ALU	15	add immediate

Table 1: Encoding of the primary opcode field

Effective channel length	0.9u
Gate oxide thickness	225 Angstrom
Min FET Width	3um
Polycide (width/space)	1.5um/1.5um
Metal 1 (width/spacing)	3u/1.5u
Metal 1 contact	1.5um X 1.5um
Metal 2 (width/spacing)	3.75um/1.5um
Metal 2 contact	1.5um X 1.5um

Table 2: Process characteristics (as drawn unless specified otherwise)

logic for instruction decode, interlock detection, pipeline control, and instruction cache refill FSM total 1% of the chip's transistors. The instruction cache contains 20.5K bits organized as

Die size	6.98mm X 8.73mm
Transistors	180K
Package	176 pin plastic PGA
Signal I/O’s	136
Power pads	32
Cycle time	40ns (typical)

Table 3: Summary of chip characteristics

128 rows and 160 columns, with a cache line size of 16 bytes. The cell is a 6T static cell. Overall access time is 1/2 cycle. The register file consists of 48 32b general purpose registers. The register cell is an 8T static cell with two dual-rail ports. Access time of the register file is 1/4 cycle.

3.1. Pipeline Design

The processor has a four-stage pipeline: instruction fetch (IF), execute (ALU), memory access (MEM), and write back (WB). The resources and timing of the pipeline are shown in Figure 4. During IF, the PC is selected and the instruction is fetched from the direct-mapped on-chip cache. The parity of the instruction is checked, the source registers required by the instruction are fetched, the instruction is decoded, and all (hardware) interlocks are computed in parallel with the instruction cache tag comparison. Note that due to the simple architecture and organization there is no need for an instruction decode stage, as is present in most machines. This saves a pipeline stage before the branch determination point in the pipeline, allowing the branch delay to be one cycle instead of two and improving the sustained performance by approximately 10% for a given peak performance.

In the ALU pipestage ALU operations, address calculation of memory access instructions, and branch address calculations are performed. For full performance many machine organizations require separate adders for branch displacements, memory addresses, and ALU operations. Since these calculations are in the same pipestage in this machine, only a single shared adder is required. This increases the utilization of the ALU to about 79% of the instructions executed on a wide range of benchmarks.

In the MEM pipestage memory access instructions send out their address over the 32b address bus at the beginning of the cycle. Load instructions receive their data at the end of the MEM cycle, but the result of the external cache tag comparison is not known until the end of the WB pipestage. The results of all instructions are written to the register file in the WB pipestage. Stores read their data out of the register file during WB as well, so the only difference to the register file between loads, ALU operations, and stores is the R/W control.

All instructions commit in their WB pipestage. Having a uniform commit point simplifies the precise handling of exceptions. Also, by delaying the write of ALU results until WB instead of writing them immediately after the ALU pipestage in MEM, a single write port to the register file is sufficient.

Insert die photo with floorplan overlay here.

Figure 3: CPU photomicrograph

Unit	Transistors	%Transistors	%Area
I-Cache	134,540	75.0	30.3
Register file	16,516	9.2	5.8
Data path	15,486	8.6	5.7
Control logic	1,757	1.0	2.8
Pads & interface	11,091	6.2	33.7
Global routing			14.7

Table 4: CPU transistors and area by unit

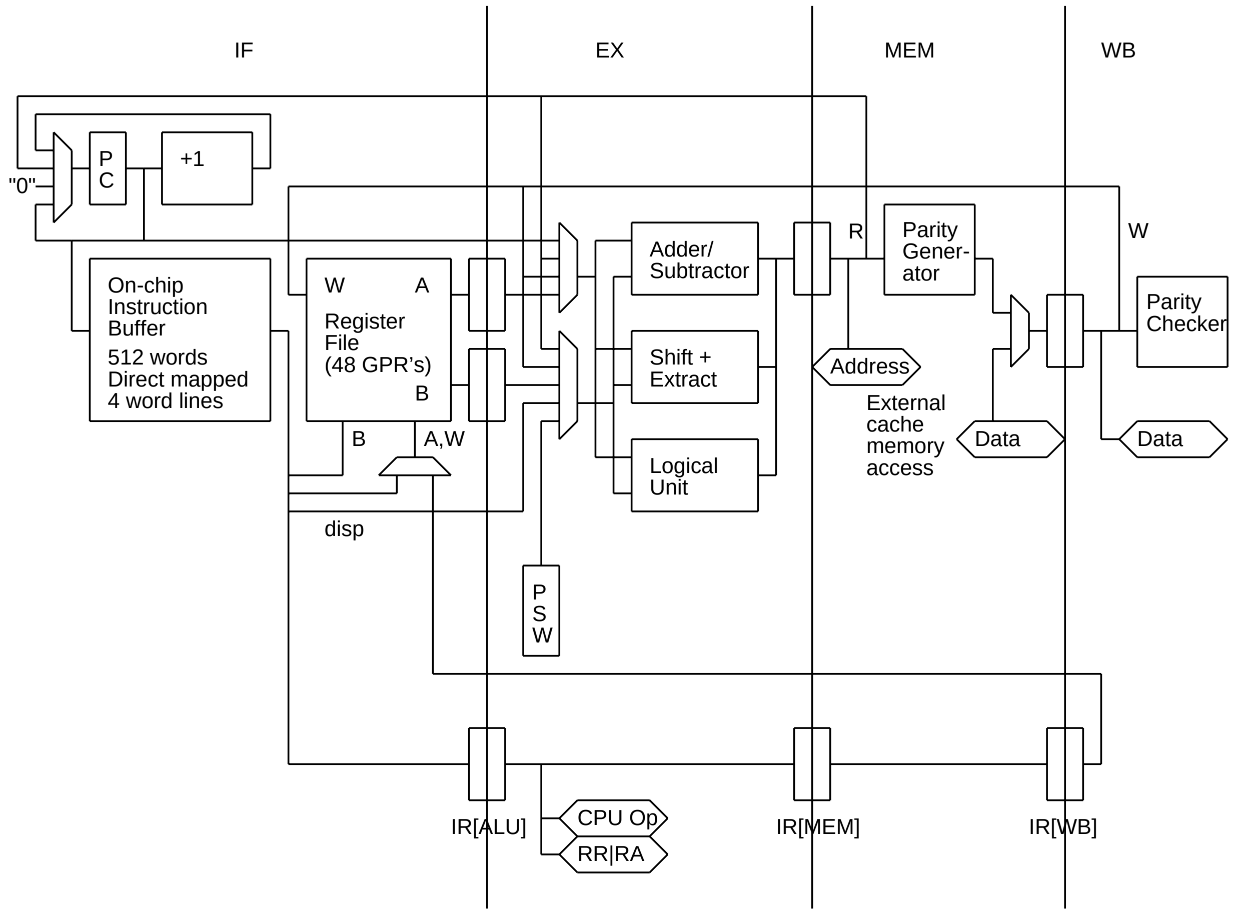


Figure 4: CPU pipeline and machine organization

Because of the very simple and regular microarchitecture, the datapath required can also be made simple and regular. One aspect of this can be seen in the bus plan of the chip (see Figure 5). Only six metal tracks are required across the width of the data path, including one track alternately used by power and ground in odd and even bits (i.e., odd bits in the datapath are mirrored). The other five tracks are generally used for the A and B source operands, the result bus R, the memory result M, and a local track used for intracell wiring. These four busses are the minimum required in the case of a load instruction followed by an ALU instruction both bypassing into another ALU instruction. Because there are only six metal 2 tracks in the datapath, the datapath width can be reduced to 36um in a 1.5um process, or only 48 lamdba. 48

lambda also was useful in that it is divisible by many numbers, allowing a wide variety of structures to be efficiently doubled, tripled, or quadrupled up in one datapath pitch.

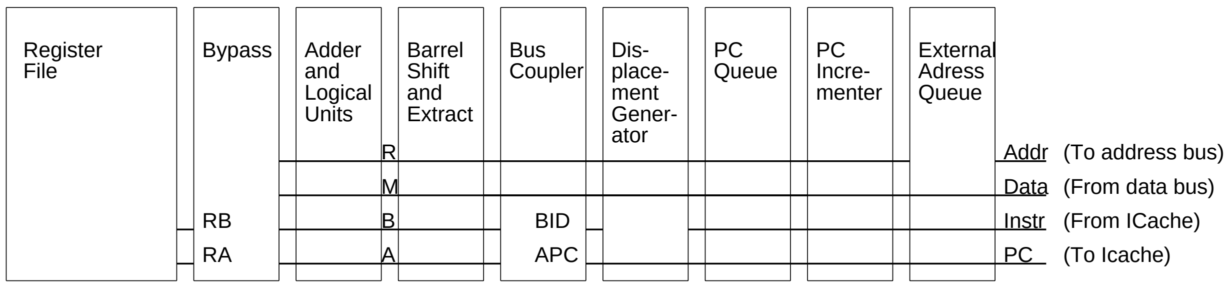


Figure 5: Bus floorplan in the CPU datapth

During the design of the machine we considered a more aggressive pipeline design that split each of the pipestages of Figure 4 into two pipestages. This would double the peak rate (bandwidth) of instruction execution, but the actual time (latency) required for the execution of instructions would not change. Unfortunately, benchmark performance tends to be limited by data dependencies and control dependencies (i.e., branches), so doubling the peak rate without improving the latency of dependencies only resulted in many more stall cycles and decreased utilization of the machine. We simulated a collection of large benchmarks using a parameterizable optimizing compilation and simulation system, on both the MultiTitan and the double-pipelined machines [6]. The average sustained performance improvement on a wide range of benchmarks from doubling the number of pipeline stages was only 18%. Given the paucity of the improvement, the increased design effort required for double-pipelining, and the cycle time risk of the extra complexity, we decided to implement a four-stage MultiTitan pipeline.

3.2. Distributed control and decode

One of the interesting aspects of the machine design was its emphasis on distributed decode and control. Figure 6 contrasts distributed and centralized control schemes. The centralized instruction decode and pipeline control is similar to that used in the Stanford MIPS processor [3] as well as typical of other processors. In a centralized decode scheme, signals must typically go from a chip-wide instruction bus through decode or control logic and fan out to the rest of the chip. In this case two chip-width crossings are required in the worst case to generate a control signal. Moreover, on the average the generation of each control signal requires the crossing of the entire chip.

In a distributed control scheme the need for global communication is much reduced. The worst case signal crossing required to generate a signal is one chip-width crossing, but many signals can be generated with almost no global communication (see Figure 6).

Note that simple instruction decode and pipeline control is a prerequisite for distributed control implementations. Machines with complex, interdependent, or irregular control structures are unlikely to be able to afford distributed control methods.

The distributed control places different requirements on the implementation structure used in comparison to centralized control or decode. Typically in centralized approaches PLA’s, ROMs,

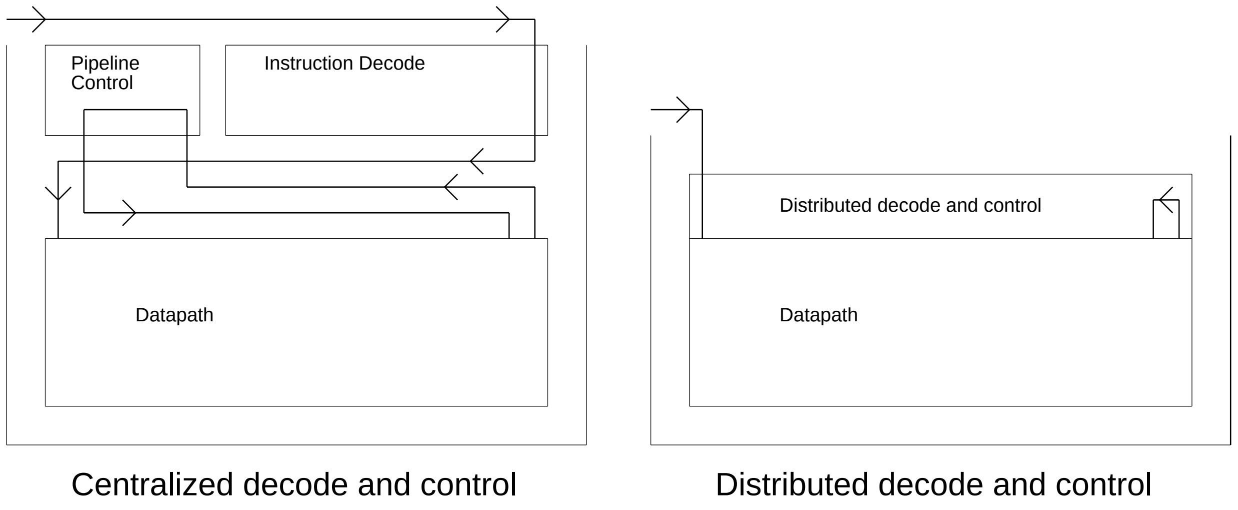

Pipeline Control
Instruction Decode
Datapath
Centralized decode and control
Distributed decode and control
Datapath
Distributed decode and control

Figure 6: Distributed vs. centralized decode and control

or other large structures are used. Optimizing CAD programs are used to share logic gates between functions within these structures in order to remove redundancy. However, not only are centralized schemes penalized by the wire delay to get to them and to get back to the part of the machine being controlled, but large amounts of area are also required for the routing of these wires. Finally, if PLAs are used to implement centralized control or decode, they are themselves dominated by wire area and wire delay.

In contrast distributed control generally avoids optimization to share logic and eliminate redundancy. For example, in the MultiTitan CPU chip if the same control signal is required at two ends of the data path, the logic to compute the control signal is merely duplicated at both ends of the data path. Not only does duplicating a few gates take up less space than bussing a control signal all the way across a chip, but computing it locally is much faster as well. As another example, instead of putting all the logic into a single large structure (e.g., PLA), small strips of logic gates are placed all along the data path to compute control signals wherever they are needed. The implementation of these logic strips is discussed in section 5.

Even in a distributed control scheme some signals must cross the width of the chip. For example, if one section of the machine identifies the need to stall, it must be able to communicate this quickly to the rest of the machine. In the MultiTitan CPU we developed a simplified method of global pipeline control. This is aided by the fact that in the MultiTitan all instructions pass through every pipestage and instructions pass through each stage in order.

Typically stalls and interrupts are communicated by a number of signals denoting exceptional conditions (e.g., cache miss, instruction buffer miss, page fault, interlock type x, etc.) These exceptional conditions must be prioritized and translated into control signals for each pipeline stage. Many of the exceptional conditions or interrupts come from distant portions of the machine and may spend a considerable amount of the cycle just traveling to a central prioritizer.

In the MultiTitan we implemented a distributed stall and interrupt control system without a central prioritizer which is very simple and very fast [5]. Each pipeline stage has associated with it a signal called "LoadX", where X is the pipeline stage name (e.g., LoadIF). Instead of signalling exceptional conditions in terms of the event (e.g., "cache miss"), exceptional conditions are signaled within the CPU by deasserting the LoadX signals required by that exception. When the pipeline control for one pipestage is deasserted, in order to prevent upstream instructions from entering the stalled pipestage, the detector of the exceptional condition must deassert all LoadX control signals for stages previous to X as well. Since all deassertions of a particular LoadX signal are wired-ORed, the net effect will be the same as the exceptional condition that deasserts the most signals, providing the same functionality as a central prioritizer, but in a distributed fashion. (The wired-OR is provided by a precharged node with multiple pull-downs, one for each stall generator.)

Given these LoadX pipeline control signals, the control of the machine is very simple. Many circuits, such as the shift control of the PC queue, connect directly to the pipeline control signals. The most complex use of the pipeline control signals requires a two-input NAND gate and an inverter.

4. Circuits

4.1. Clock

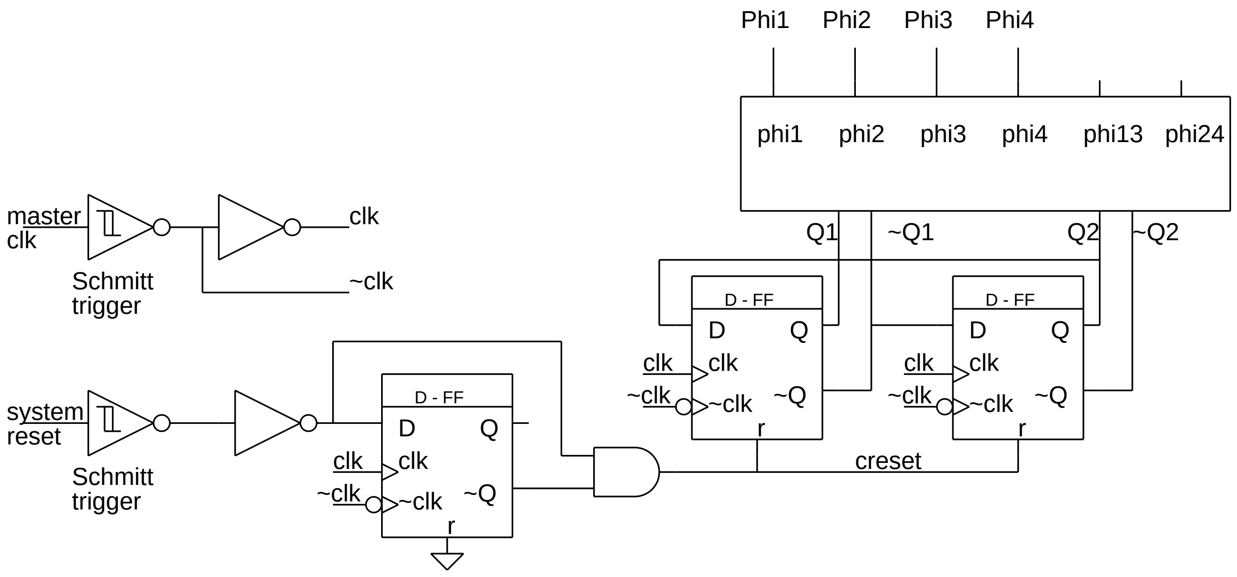
The CPU uses a four phase clocking scheme. The schematic of the clock phase generator is shown in Figure 7. In the nominal case, the phases are derived from the 100 MHz external master clock. An external system reset synchronizes the clocks on all the chips in the system. In practice, the incoming master clock at best looks like a sine wave with noises. To turn the sloppy wave form into sharp edges, a Schmitt trigger receiver is used to receive both the master clock and the system reset. A reset pulse, creset, is generated by a D-FF(D-Flip Flop) and an AND gate. It resets the D-FF's of the phase generator. The timing of the system reset is not critical. It can either lead or lag the master clock by more than 90 degrees without causing errors. On board delay lines are used for fine tuning.

The clock generator consists of a gray-code counter and a phase generator/driver. The gray-code counter divides the input clock by four and generates Q1(Phi34) and Q2(Phi23) that are offset by 90 degrees. Both true and complement of Q1 and Q2 are then used to generate the desired phases with simple logic. The same clock phase generator is also used in the cache control unit and the floating point unit with only the phase drivers tuned to their respective loadings. On the CPU, only simple phases are used. Note that in our scheme, only the falling edge of the master clock is used to trigger the phases. Clock skews caused by the imbalance between the rising and falling edges are avoided.

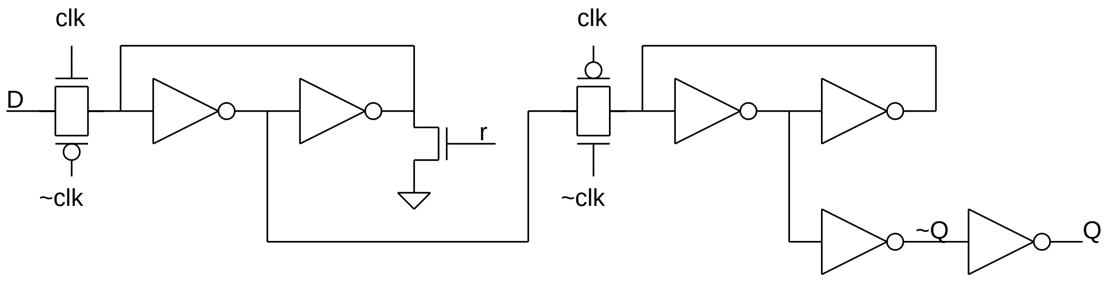
Clock phases are also driven off the chip with near-by pad drivers. Aside from testing purposes, they could be used to match the chip set and minimize overall system clock skews. The ultimate matching is to fabricate the chip set on the same wafer to minimize process variations.

4.2. The Register File Sense Amplifier

The register file consists of 48 32-bit registers. The register cell is an 8T static cell with two dual-rail ports. The access time is one phase with one write access in phase 2 of the WB pipe-stage and two read accesses in phase 4 of the IF pipe-stage. As shown in Figure 8, the 3-bit pre-decoding(not shown) is already finished in the pre-charge phase(phase 1 or 3). In the



(A) Clock Phase Generator



(B) D-Flip Flop

Figure 7: Schematic of the on-chip clock generator

evaluation phase (phase 2 or 4), the final decoding is done followed by sensing of the bit lines and driving of the datapath busses. Since this all happens within one phase, a timing chain is built to gate the sense amplifier. Four eight-way NANDs are used to generate the "done" signals for the final decoding stage of both ports. The "done" signals are then AND together with the evaluating clock phase followed by an inverter delay chain. The delay chain is tuned such that the sense amplifier is turned on when sufficient differential voltages have developed on the bit lines to guarantee correctness.

It turned out that quite enough margin exists in the evaluation phase. It could probably have been done without the sense amplifier.

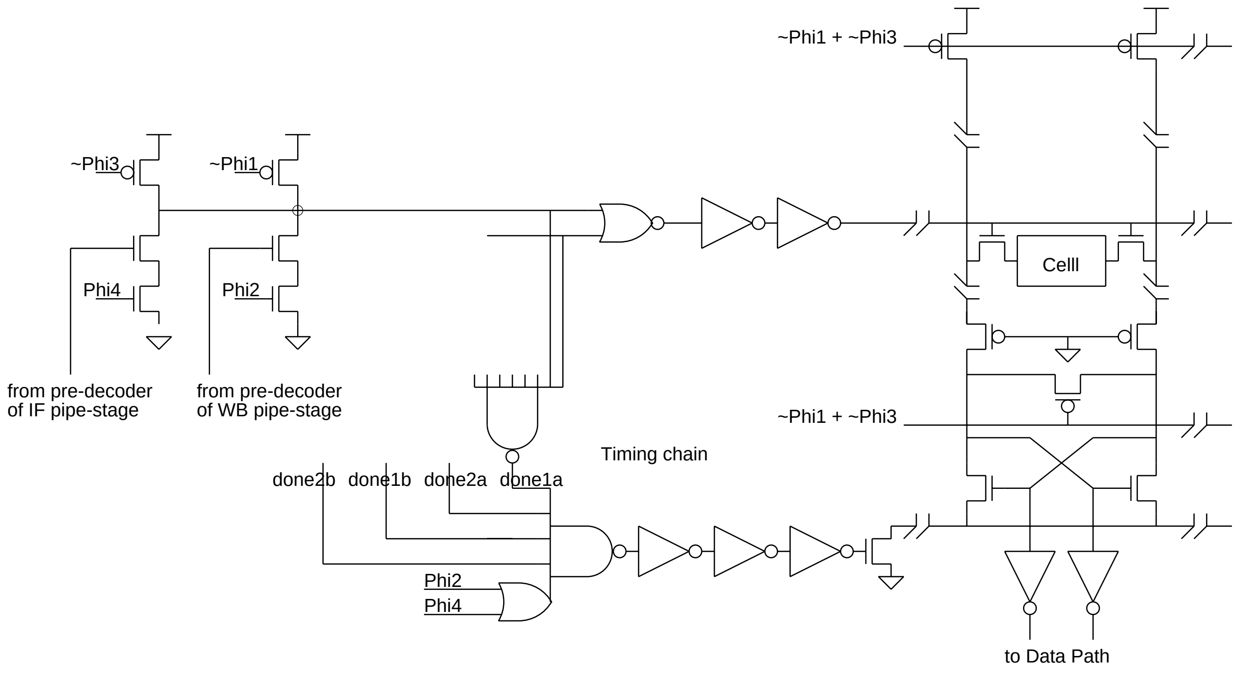


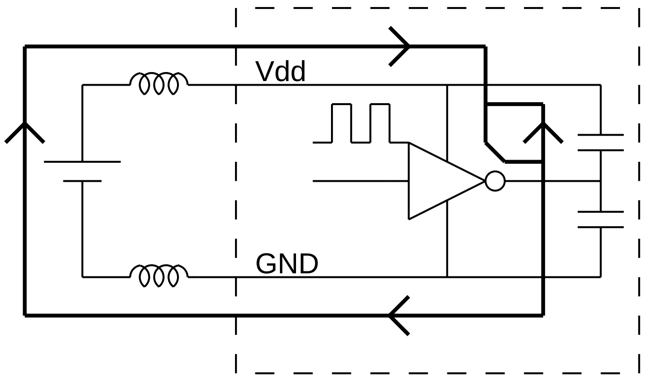
Figure 8: Schematic of the register file timing chain

4.3. Issues of Ldi/dt and Decoupling Capacitors

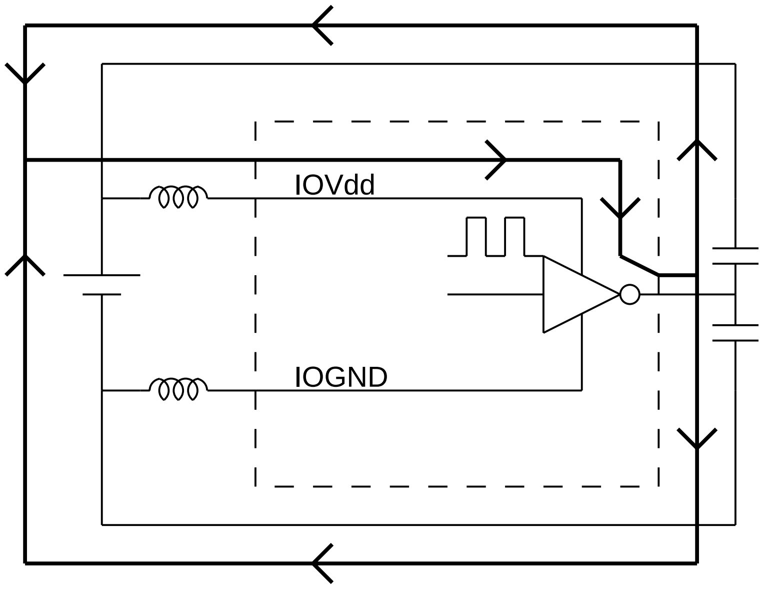
Ldi/dt problems arise mainly from the lead inductance of the chip package. Noises are generated when pad drivers are switched. Internal Vdd and GND are separated from that of the pad drivers to decouple the IO switching noise. Figure 9 shows a simple schematic of an inverter driving an on-chip (A) and an off-chip (B) capacitor. Dashed lines depict the chip boundary and thick lines show the current path. It is obvious that the lead inductances of IOVdd and IOGND cannot be bypassed by either an on-chip or an off-chip decoupling capacitor. Reducing di/dt and/or the package inductances is the only means to cut down the output switching noise. On the other hand, the di/dt generated by the internal circuits can be bypassed through a decoupling capacitor connected between Vdd and GND.

On the 176-pin PGA that was used, 3 IOVdd and 4 IOGnd are allocated on one side of the chip. These are double-size pads that are doubly bonded to reduce the resistances and inductances. In order to minimize the effective number of simultaneously switched drivers, the address pins and the data pins are interleaved and distributed uniformly along the four sides. With this scheme, the effective number of simultaneously switched drivers is about halved. It is estimated that the effective inductance one pad driver sees is about 8nH on the IOVdd lead and 7nH on the IOGnd lead. The pad drivers are designed to these parameters with the worse case on-board trace. In the typical case, the simulated chip-crossing delay is about 5.5 ns.

The four triple-size corner pads are used for internal Vdd and GND. As mentioned earlier, di/dt generated by internal circuits can be bypassed through a decoupling capacitor. In fact, in the CMOS n-well technology, a significant amount of the capacitance comes naturally from the capacitance between the n-well and the substrate. Consider the simple model shown in Figure 10.



(A) On-chip load



(B) Off-chip load

Figure 9: The up-going current path of on-chip and off-chip capacitive loadings

In this simple model, only the chip package inductances, the non-pre-charge type circuitry, e.g. the clock driver, and the pre-charge type circuitry are differentiated. From direct layout extraction, the total capacitance between Vdd and GND comes to about 2000 pF (C_{well} ~ 45% and C_p ~ 55%) and the ratio of C_{gnd} to C_{vdd} is > 3. The time constant of a well is given by $l^2 r_{sheet} c_{unit\_area}$, where l is the largest distance from the far point of the well to the well contact, r_{sheet} is the well sheet resistance and $c_{unit\ area}$ is the well capacitance per unit area. In our CMOS technolgy, $r_{sheet} c_{unit\_area}$ comes to about a quarter of a pico second for area in um^2. Typically, l is less than 60 um, i.e. R_{well}*C_{well} is under a nano second. This is equivalent to say that for switching events with time constants greater than a nano second, C_{well} is effective for decoupling purposes.

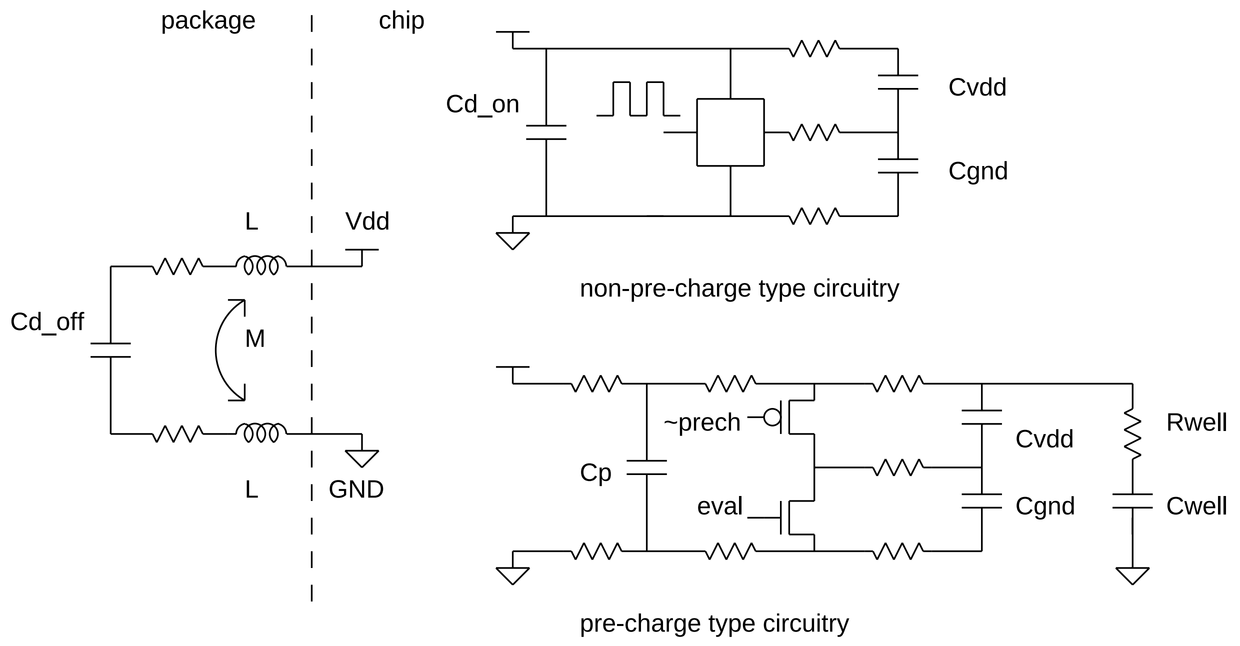


Figure 10: The pre-charge type and non-pre-charge type circuitries.

There's one very interesting aspect of the pre-charge circuitry. Since pre-charge always takes a full phase, it has a relatively long time constant(> 2 ns). Note also that in the pre-charge phase, the current flows from Vdd through the pre-charge transistors, C_{gnd} and then to GND. C_{vdd} discharges through a local loop. In the evaluation phase, C_{gnd} discharges promptly through a local loop, while only the small C_{vdd} requires charging current from the power supply. This greatly alleviates the Ldi/dt problem in that the larger capacitance is always switched very slowly ($C_{gnd}/C_{Vdd} > 3$). Even in the case where C_{Vdd} and C_{gnd} are comparable, only half of the capacitive loading is subject to fast switching.

Indeed, most of the heavy loadings in the design occur in pre-charge type circuitry. Good examples are the bit lines, the busses, and the domino circuits. Furthermore, in one cycle, only one third of the columns in the instruction cache are active, and different circuits are pre-charged at different phases. Within a phase, the effective capacitive loading that requires current from the power supply is much smaller than the apparent capacitive loading. It is estimated that the capacitances from C_{well} and C_p is about enough to effectively decouple the lead inductances in the package to ensure a less than 10 percent of the internal power supply variations due to internal di/dt.

To effectively decouple the clock circuitry, an on-chip decoupling capacitor using thin oxide gate capacitors was designed. It is built with narrow stripes to ensure a fast response time (< 0.5ns). The value of the capacitance is about 800 pF which is more than 10 times larger than the clock phase loadings. Active pre-charged nodes fall at a slower rate than that of the clock phases and are well decoupled by the well to substrate capacitance already. The decoupling capacitor is not limited by the available silicon. But building larger capacitors will be of marginal utility and sacrifices the chip yield.

5. Design, Verification, and Test Methodology

In the design of the MultiTitan CPU we used a sophisticated yet conservative design methodology. In general the circuit designs were kept simple and regular. The most important advantage of this was that the simple and regular circuits could be fully understood by high-level chip-wide analysis tools. This allowed us to optimize the design as a whole, and saved needless microoptimization of areas which were not critical in the design as a whole. For example, we only used SPICE on six circuits in the machine: the register file sense amp, the instruction cache sense amp, the master clock, pad di/dt, the decoupling capacitor, and the adder carry chain. Even though we did run SPICE on these circuits, they were still relatively simple and could be understood by our switch-level simulator and timing verifier. This allowed us to run these tools on the whole chip without having to remove "tricky circuits" that the tools could not understand. This is important because design bugs often result from parts of a chip that are too tricky or too large to be modelled directly, and have to be replaced by simpler circuits in the simulation. This simplified modeling then can easily obscure design problems, especially problems in the interface area between the models.

By being able to simulate test vectors directly from the pads of the complete chip during simulation, including the 100MHz AC clock input, we achieved unparalleled confidence in chip functionality. This was validated when we tested the CPU chip on our test board after fab. Once the clock was fed in properly and the rest of the board was debugged, large programs immediately began to run without any modification of the object code whatsoever. As of this time no design errors have been found, not even any requiring the "redefinition" of any instruction behavior.

This high-level design approach also was very efficient in terms of design time required. Only 2.5 man-years were required for the design, including all aspects of design from the definition of the instruction set, through the architecture, organization, logic, circuit, layout, and testing.

5.1. Design process overview

The design process was started by prototyping typical circuits (adders, register files, etc.) in the target technology. These initial designs provided data for making architectural and organizational tradeoffs in the machine. Each design was optimized locally (e.g., not all had the same data path pitches), so they were all thrown away when the real logic, circuit, and layout designs were begun.

Concurrent with the technology evaluation, an instruction-level simulator and the code generation phase of our compilation system [15] was retargeted for the proposed architecture. The simulator was later expanded to model the pipeline and memory subsystem at a high behavioral level. Doing the design of the architecture and organization of the machine at the same time as the technology evaluation enabled tradeoffs to be made between all these disciplines, resulting in a simpler and faster design. Moreover, only in the context of having code available from compilers can large programs be realistically simulated. This permits the sustained system performance to be optimized, instead of optimizing the peak performance of the chip to the detriment of system performance, as can occur when basing design decisions on small hand-written code sequences.

Next the detailed design of CPU was begun. The Magic layout system [8] was used for the layout editing, design rule checking, and extraction. It was extended locally to check electrical well design rules.

Next a detailed functional-level simulator was written in Modula-2. The functional-level simulator included modules for the CCU and FPU chips as well as the memory system interface. Unfortunately this was not developed until the detailed design of the CPU was nearly complete. A number of changes to the control logic were found to be necessary as a result of the functional simulations in order for the CPU to operate properly in a systems environment when handling interrupts and stalls.

The functional-level simulator was also used to generate test vectors for switch-level simulation. The same test cases that were run on the functional-level simulator could then be run on the switch-level simulation of the chip, albeit 1,000 times more slowly. RSIM [14] was used for switch-level simulation. One of the major reasons for choosing it was that it performed timing-based switch-level simulation; this was necessary for circuits like the register file sense amp timing chain.

A much improved version of TV [4] was used for static checking, timing analysis, and clocking verification. The static checking was used to verify that our simple and regular circuit design methodology was followed. This included the following checks:

- No dynamic nodes driving dynamic nodes through pull-down transistors. (See Figure 11.) This ensured a high margin on dynamic nodes.
- Transitioning coupling capacitance to dynamic nodes resulted in signal deviations within noise margin limits.
- Charge sharing in dynamic structures, such as within evaluating domino blocks, was within noise margin limits.

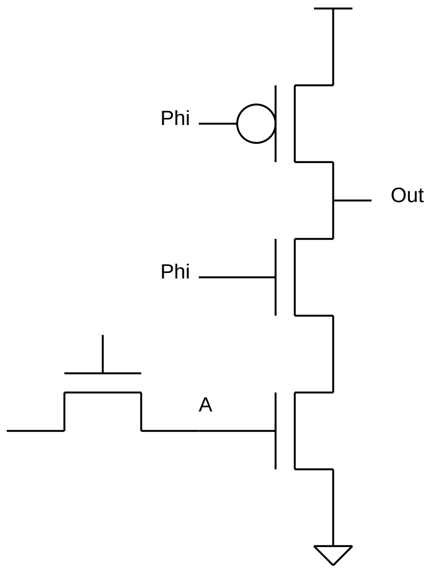


Figure 11: Dynamic logic driving dynamic logic via pull-down

Finally, Ariel [13] was used for power drop simulation of Vdd and GND networks. It performed its analysis by extracting the parasitic resistors in the Vdd and GND network. Then it took a trace of switching events from RSIM, and played the current generated from these events across the power network. Based on this power network simulation, several roundabout connections of Vdd and GND to the control logic were rerouted into more direct connections, saving several tenths of a volt drop in the worst case.

5.2. Standard cell implementation of distributed decode and control

As mentioned previously, the decode and control logic of the machine was distributed adjacent to its use, and was designed using a standard cell model. We did not have any suitable standard-cell tools available for the design, so since there was only a small number of gates required in total, and the relative placement of the gates was defined by their use in the data path, the layout of the standard cells into layout strips was performed by hand. Part of one of these control strips is given in Figure 12. Because of the simplicity of the instruction encoding, at most two levels of logic and an inverter are required to generate all the decoded instruction control signals in the chip. The two levels of logic and optional inverter correspond to the leftmost three columns, with the inverters in the leftmost column. Because all the instruction decode is performed in the IF pipestage, if control signals are required by later pipestages then they are delayed by the appropriate number of pipestage delay blocks. These delay elements are placed in the the next three columns. Finally, in the rightmost column one additional level of logic is provided for combining the outputs of signals delayed by different amounts.

5.3. Clock tuning with TV

Extensive clock tuning was performed with TV. This tuning consisted of two major aspects: tuning for functionality (i.e., clocking verification), and tuning for performance (i.e., timing analysis).

The performance of the overall design was improved by about 50% through the use of timing analysis. However, this performance improvement is actually the least important implication of the timing analysis. Presumably something very close to the final performance could have been achieved without timing analysis but with large amounts of simulation and careful hand analysis. However, the most important benefit of the timing analysis was the ability to do a simple straightforward circuit design everywhere and to only put in significant design effort at improving performance in the places where it was really necessary in the global framework.

More important than performance is correct functionality. Correct clocking was assured by the clocking verification provided by TV. In clocking verification the set-up and hold times of all the latches and precharged nodes in the machine are verified to have a specified timing margin. Although simulation-based approaches can find some outright set-up or hold violations, they cannot find low margin latches with a practical amount of designer effort.

In the first design pass control lines in the chip were first buffered based on their capacitance. The buffers for control lines were built in multiples of a minimum size. A command was added to Magic which extracted the capacitance of a node interactively; this was used to choose the proper number of buffer sections for each control line. Then based on the timing margins computed by TV, the various control lines were either skewed forward or backward in time. More

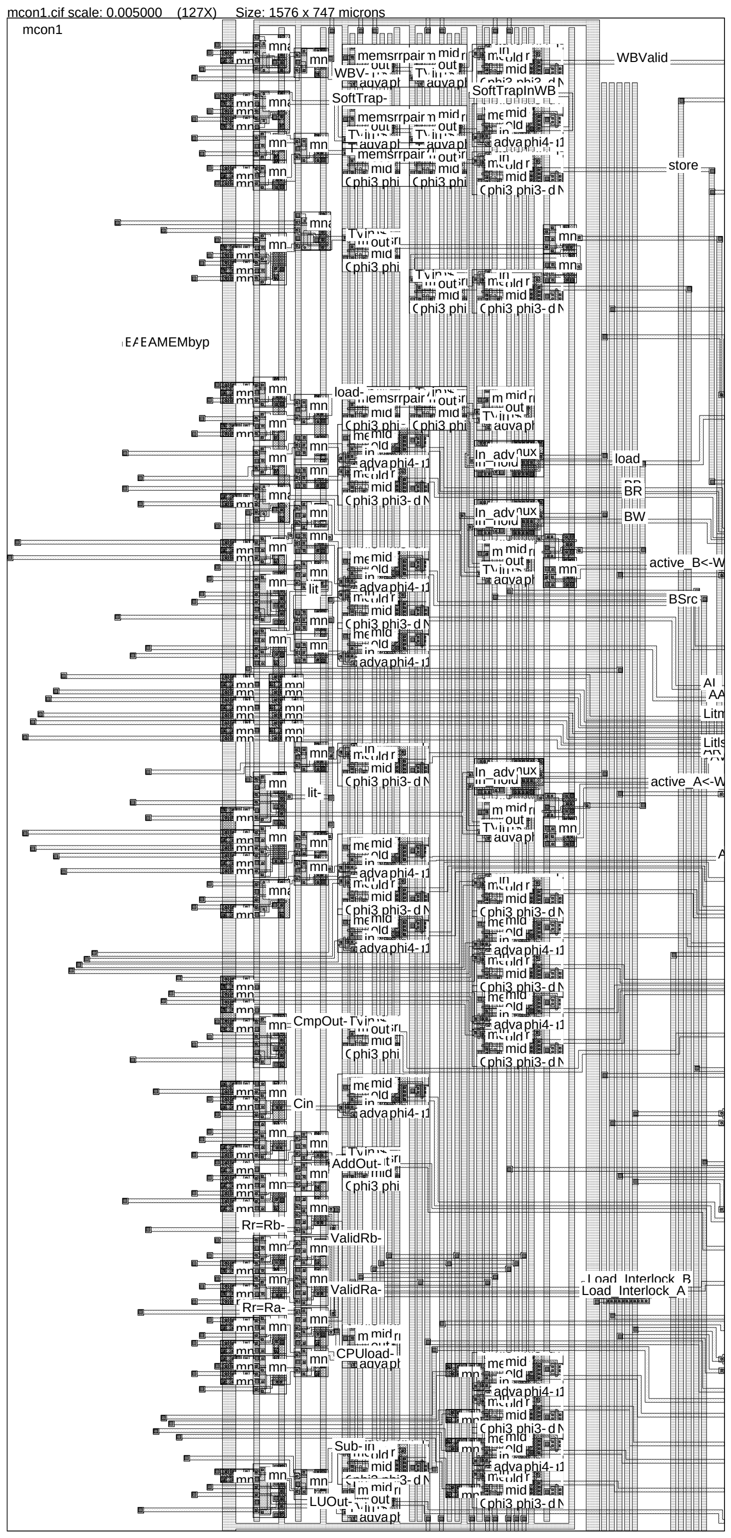


Figure 12: Standard cell decode and control trays

buffer sections were added in parallel to the already existing buffers for signals which needed to be advanced in time to improve margins. For signals which needed to be slowed down, buffer sections were removed or a pair of minimum size buffers were added in series with the control signal. The net result was that several nanoseconds of overlap between adjacent clock phases could be safely tolerated by the chip, even with a 10ns target time per phase.

6. Performance

The CPU was sent out for fabrication in October 1987 and one lot came back in April 1988. The chip was designed for typical operation with a 40ns cycle time and a 100Mhz clock. The CPU was tested on an evaluation board with 512KB of fast RAM. This board resided on the I/O bus of our ECL Titan computer. The maximum operating frequency of the parts returned ranged from 70Mhz to 90Mhz. This performance was consistent with slower than typical processing parameters. At their maximum operating frequency large programs ran on the CPU chip without modification of their object code. At frequencies even slightly above this (e.g., <10%), even the shortest test cases stopped working. Since the chip was tuned with TV so that everything was of similar criticality, having everything fail at about the same point is a confirmation of the tuning performed with TV.

The average contribution of various factors to the simulated system CPI when executing the C compiler, yacc, Livermore loops (scalar code), Linpack (scalar), whetstones, and the Hennessy Stanford benchmarks is given in Table 5. (This data is for simulated system performance, since the evaulation board did not contain the FPU and CCU chips.) Combined with the performance data, this yields a projected sustained system performance of 18 MIPS. It is expected that with typical processing 20 sustained MIPS in a system environment will be achieved.

Contribution	Cycle penalty	CPI burden
Normal instruction issue	1	1.000
NOP instructions	1	0.024
On-chip I-cache misses	2	0.032
Load interlocks	1	0.064
Store and transfer interlocks	1	0.074
Wait for result of *,/, and FP	varies	0.056
External 128KB direct-mapped 32B line cache misses	10	0.023
Total		1.273

Table 5: Sustained average cycles per instruction

7. Summary

In this paper we described the design of a CPU chip with a high ratio of sustained system to peak performance (0.80). Attaining a high ratio of sustained system performance to peak performance avoided wasting circuit design effort at an architectural level by inefficiently using the machine resources required. By keeping the design simple and regular both at an architectural and circuit level, and by using high-level tools on the complete design, a high sustained performance (20MIPS) was obtained with relatively little design effort (2.5 man years).

8. Acknowledgements

David Boggs designed the CPU evaluation board and produced the speed test results. Bob Alverson, Scott Nettles, and Don Stark provided CAD tools instrumental in the design of the CPU. Jeremy Dion developed the functional simulator of the CPU. David Wall provided the retargetable compilation and instruction-level simulation system used in architecture studies. Leon Yang, Mark Johnson, Bruce Wooley, and the reviewers provided valuable comments on an early draft of this paper.

References

[1] Dobberpuhl, D., et. al.
A 32b Microprocessor with On-Chip Virtual Memory Management.
In *The 1984 International Solid-State Circuits Conference*, pages 178-179. IEEE Solid State Circuits Council and the University of Pennsylvania, February, 1984.

[2] Fisher, A., et.al.
CMOS Implementation of a 32b Computer.
In *The 1989 International Solid-State Circuits Conference*, pages 80-81. IEEE Solid State Circuits Council and the University of Pennsylvania, February, 1989.

[3] Hennessy, J. L., Jouppi, N. P., Przybylski, S., Rowen, C., and Gross, T.
Design of a High Performance VLSI Processor.
In Bryant, Randal (editor), *Third Caltech Conference on VLSI,* pages 33-54. Computer Science Press, March, 1983.

[4] Jouppi, Norman P.
Timing Analysis and Performance Improvement of MOS VLSI Designs.
In *IEEE Transactions on CAD*, pages 650-665. July, 1987.

[5] Jouppi, Norman P., Dion, Jeremy, Boggs, David, and Nielsen, Michael J. K.
MultiTitan: Four Architecture Papers.
Technical Report 87/8, Digital Equipment Corporation Western Research Lab, April, 1988.

[6] Jouppi, Norman P., Wall, David W.
Instruction-Level Parallelism for Superscalar and Superpipelined Machines.
In *The Third Symposium on Architectural Support for Programming Languages and Operating Systems*, pages 272-282. IEEE, April, 1989.

[7] Nielsen, M. J. K.
Titan System Manual.
Technical Report 86/1, Digital Equipment Corporation Western Research Lab, September, 1986.

[8] Ousterhout, John K., Hamachi, Gordon T., Mayo, Robert N., Scott, Walter S., and Taylor, George S.
The Magic Layout System.
IEEE Design and Test of Computers :19-30, February, 1985.

[9] Perry, Tekla S.
Intel's Secret is Out.
Spectrum :22-28, April, 1989.

[10] Rowen, C., et. al.
RISC VLSI Design for System-Level Performance.
VLSI Systems Design :81-88, March, 1986.

[11] Rubenfeld, P., et. al.
A 32b Microprocessor with On-chip Instruction and Data Caching and Memory Management.
In *The 1987 International Solid-State Circuits Conference*, pages 32-33. IEEE Solid State Circuits Council and the University of Pennsylvania, February, 1987.

[12] Sachs, H., and Hollingsworth, W.
A High Performance 846,000 Transistor Unix Engine: The Fairchild Clipper.
In *Proceedings IEEE International Conference on Computer Design: VLSI in Computers*, pages 342-346. IEEE Computer Society Press, October, 1985.

[13] Stark, Don, and Horowitz, Mark.
Analyzing CMOS Power Supply Networks using Ariel.
In *The Proceedings of the 25th Design Automation Conference*, pages 460-464. IEEE/ACM, June, 1988.

[14] Terman, C. J.
Simulation Tools for Digital LSI Design.
PhD thesis, MIT Department of Electrical Engineering and Computer Science, 1983.

[15] Wall, David W., and Powell, Michael L.
The Mahler Experience: Using an Intermediate Language as the Machine Description.
In *Second International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 100-104. IEEE Computer Society Press, October, 1987.

WRL Research Reports

WRL Technical Notes

Table of Contents

List of Figures

List of Tables