The Architecture of the P1

A 250MHz SPARC in GaAs

Pete Wilson
May 1989

Prisma P1 Goals

Hot SPARC UNIX Box
250 MIP, 100 MFLOP peak
15-20x integer performance of Sun 4/280
Gigabytes of physical memory
Massive IO capability
Networkable "black box"
Reasonable price

Mainframe-Quality Software
"Real Computer" system administration tools
OS enhancements - thousands of users

Out-of-the-Box Software
"All" user-mode Sun 4/280 and SPARC ABI binaries run

Adopt standards wherever possible
Technology

Processor Devices
sub micron GaAs, 3 metal levels, ~100ps gate delays
112 pin chips, 200 x 160 mil
1-8 W power dissipation
Prisma macrocell design, Gigabit foundry

Remainder
ECL IO interfaces, secondary cache
Generic silicon DRAM main memory

Design
Chips: Cadence
Boards: Cadnetix
Logic: Verilog

Technical Overview
Constraints
- SPARC ABI/Sun binary compatible
- performance no worse than ratio of clock speeds (wrt Sun4)
- implementable with low risk
- much higher ratio of CPU speed to memory chip speed
- much faster IO needed
- commitment to industry standards
- technology forces non-sequential machine

Approach
- separate I and D caches
- internal concurrency
- shorter pipe
- bigger 2-level MMU
- higher performance FPU
- multi-channel DMA-driven IO
P1 Design Approach

Very fast simple pipelined integer unit
250 MHz SPARC implementation
Short pipeline

Very high bandwidth memory system
4 Gigabytes/second bandwidth sustained
Cache hierarchy, large MMU 256 contexts
Interleaved main memory

Fast FPU
32/64 IEEE SPARC FPU superset
100Mflop sustained
Much concurrency

Very high bandwidth fast IO system
4 Gigabyte/second aggregate bandwidth
Channel-oriented IO with memory-mapped facilities
32 channels each 200Mbyte/second

P1 Integer Pipeline

Pipeline design centered on ALU and register bank needs

Registers
2 nsec cycle time

ALU
4 nsec cycle time

to get one instruction per clock, we must pipeline register read, alu operation and register write
**ALU-Register Pipeline**

Pipeline design centered on ALU and register bank needs

In the second half of a clock, we read two values from the register banks. In the next clock, these values are operated on by the ALU. In the first half of the next clock, the result is written to the register bank.

**Instruction Decode**

We use the first half of the RegRead clock to do Decode

At the beginning of each clock, decode gets an instruction from the icache. At the end of each clock, it sends the address of the instruction it will need in 2 clocks.
Scoreboarding

Because the machine can create and service multiple simultaneously active memory requests, we scoreboard the integer registers.

```
loop:
  ldsb (r10 + r20), r14
  ldsb (r11 + r21), r15
  add r10, 1, r10
  subcc r14, r15, r0
  bne loop
  add r11, 1, r11

Software complete by fault
```

Allowing memory loads to proceed in the background means that the relatively long-latency memory activity can be overlapped with other useful activity. To keep data consistent, the destination register of a load is claimed, and all the registers used in an instruction are checked for availability. This is useful even with high hit rates, given good instruction scheduling.

Secondary Cache

Main memory built from affordable devices is much too slow to be driven directly by a 4nsec clock-rate machine. So we interpose a secondary cache.

The secondary cache is a 4-way interleaved design; each bank can be active simultaneously. An Arbitrator between the MMU and the cache banks also shares cache bandwidth between CPU and IO traffic. A collector behind the cache banks provides data in the correct order to the requestors.
Data Cache Pipeline

We cannot build a single-cycle data cache with available RAMS. Thus the cache is a 2-cycle pipelined lookup.

To reduce the latency caused by the 2-stage cache pipe, results are forwarded to the ALU in parallel with being written to the registers. This provides performance similar to a single cycle cache at the cost of design complexity.

MMU

We use a 2-level MMU which closely matches the functionality of the Sun Reference MMU.

The MMU accepts requests from the Icache and Dcache. Dcache has absolute priority. MMU produces a 36 bit physical address from the 32 bit virtual address. A queue of requests is built by the MMU back to each cache. MMU is cache-like, walks own 2-level page tables. An 8-bit context number is hashed with the virtual address.
Instruction Cache

Memory is too slow to provide an instruction every 4nsec; we use an instruction cache. The cache is 16Kbyte, direct-mapped, with programmable (128-512 bit) block size and 128 bit line size.

The icache provides two instructions at a time to the Decode stage. On a miss, it can queue up many requests to the memory system.

Data Cache

Memory is too slow for loads and stores, too. So we have a data cache. The cache is direct-mapped, and 16Kb in size, with 16 byte line and block length and a writeback default policy.

The data cache lives in the pipe after the ALU. One cycle is needed for lookup, and another for a store. On a load, DC provides its value to a second Register Write stage. On a store, the normal RegWrite stage sources a value.
Memory and IO Arbiter

The Arbiter handles the interleaving of requests from CPU to memory and to IO, and of requests from IO to memory.

Main Memory

Each bank of the secondary cache has an interleaved main memory system attached to it.

Because the main memory banks are only ever accessed by the secondary cache, advantage may be taken of the relatively long transfers needed to read and write its cache lines. This allows the use of DRAM with little performance penalty.
Floating Point Unit

The Prisma FPU provides greater concurrency than the Sun design and offers 100Mflops peak (64bit IEEE).

![Diagram of FPU and IU](image)

The FPU runs in parallel with the IU. A queue carries Floating Point Instructions to the FPU; an Operand Queue carries data values from the data cache to the FPU.

Floating Point Unit

The FPU also has internal concurrency. Registers are scoreboarded for every operation. FUs are scheduled.

![Diagram of FPU and IU](image)

\[\text{multiply:} 3,7 \quad \text{divide:} \quad 15,30 \quad \text{add:} \quad 5\]
Input/Output

The P1's IO system is based around multiple autonomous DMA channels, driving HSC interfaces to multiple VME crates. Generic memory-mapped IO is available for vanilla applications.

Device Interrupts

Interrupts are queued up in the IOBus InArbiter. 1024 interrupts in each of the 15 SPARC levels may be queued, each with a 64 bit Interrupt Identifier.

Interrupts are dequeued by memory references to the Interrupt Controller.
Problems

Technology

some interesting problems that needed working through
device density is low - spread-out design
board density high - poor selection of autorouters
proprietary logic family - few design tools
interesting thermal issues

UNIX

some real problems
synchronous uniprocessor assumptions
single-threaded kernel
user trap handler assumptions

SPARC

to our surprise, no real problems