Design of the HP PA 7200 CPU

The PA 7200 processor chip is specifically designed to give enhanced performance in a four-way multiprocessor system without additional interface circuits. It has a new data cache organization, a prefetching mechanism, and two integer ALUs for general integer superscalar execution.

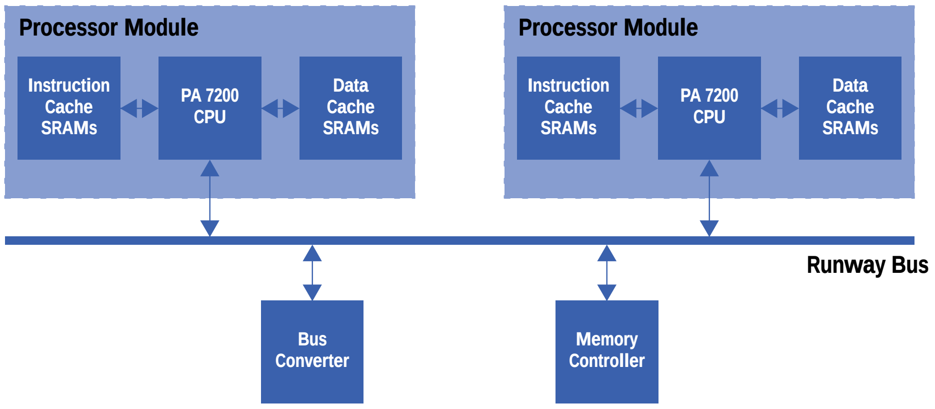
by Kenneth K. Chan, Cyrus C. Hay, John R. Keller, Gordon P. Kurpanek, Francis X. Schumacher, and Jason Zheng

Since 1986, Hewlett-Packard has designed PA-RISC[1,2] processors for its technical workstations and servers, commercial servers, and large multiprocessor transaction processing machines.[3-9] The PA 7200 processor chip is an evolution of the high-performance single-chip superscalar PA 7100 design.

The PA 7200 incorporates a number of enhancements specifically designed for a glueless four-way multiprocessor system with increased performance on both technical and commercial applications.[10-11] On the chip is a multiprocessor system bus interface which connects directly to the Runway bus described in *Article 2*. The PA 7200 also has a new data cache organization, a prefetching mechanism, and two integer ALUs for general integer superscalar execution. The PA 7200 artwork was scaled down from the PA 7100's 0.8-micrometer HP CMOS26B process for fabrication in a 0.55-micrometer HP CMOS14A process.

Fig. 1 shows the PA 7200 in a typical symmetric multiprocessor system configuration and Fig. 2 is a block diagram of the PA 7200.

Fig. 1. *The PA 7200 processor in a typical symmetric multiprocessor system configuration.*



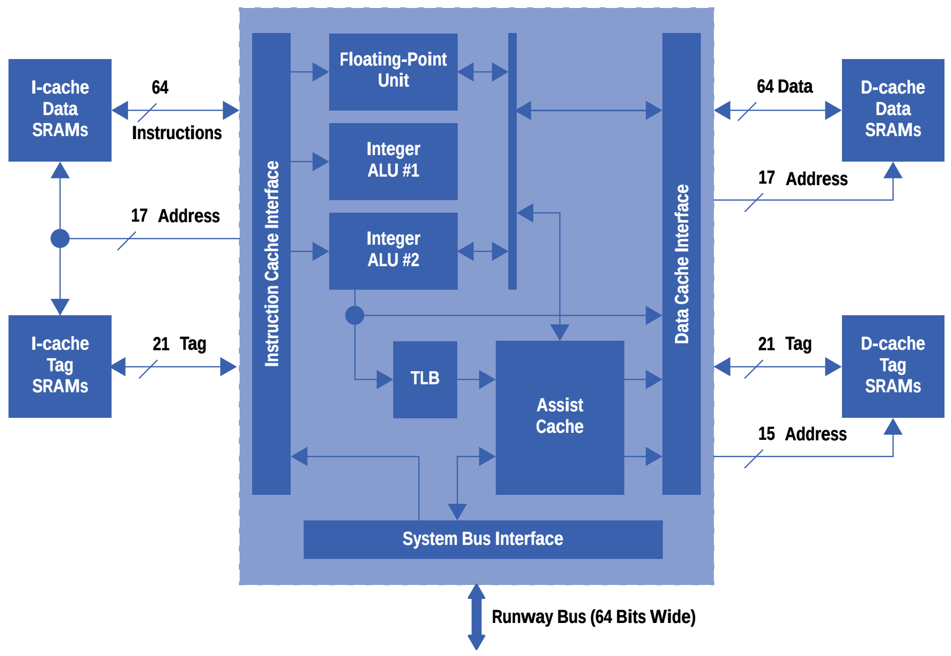
Processor Overview

The PA 7200 VLSI chip contains all of the circuits for one processor in a multiprocessor system except for external cache arrays. This includes integer and floating-point execution units, a 120-entry fully associative translation lookaside buffer (TLB) with 16-block translation entries and hardware TLB miss support, off-chip instruction and data cache interfaces for up to 2M bytes of off-chip cache, an assist cache, and a system bus interface. The floating-point unit in the PA 7200 is the same as that in the PA 7100 and retains the PA 7100's 2-cycle latency and fully pipelined execution of single and double-precision add, subtract, multiply, FMPYADD, and FMPYSUB instructions. The instruction cache interface and integer unit are enhanced for superscalar execution of integer instruction pairs. The bus interface and the assist cache are completely new designs for the PA 7200.

In addition to the performance features, the PA 7200 contains several new architectural features for specialized applications:

- Little endian data format support on a per-process basis
- Support for uncacheable memory pages
- Increased memory page protection ID (PID) size
- Load/store "spatial locality only" cache hint
- Coherent I/O support.

Fig. 2. *Block diagram of the PA 7200 CPU.*

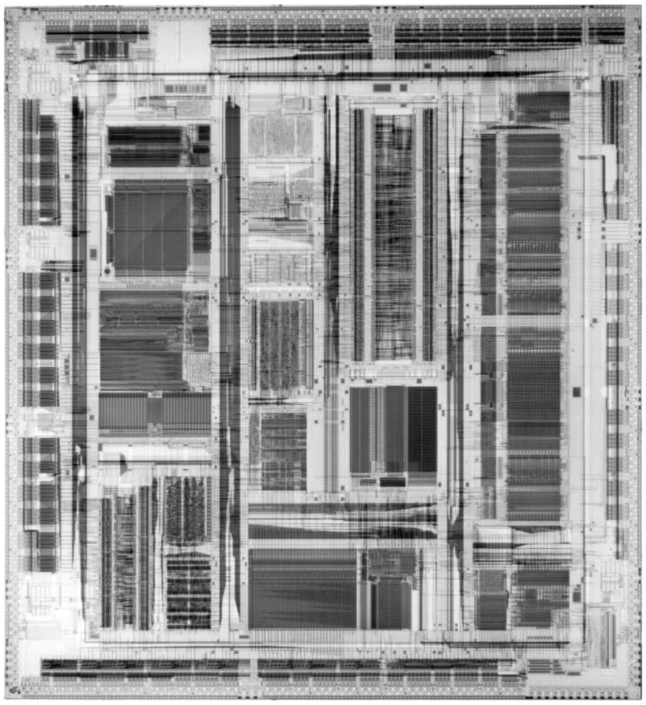


The CPU is fabricated in Hewlett-Packard's CMOS14A process with 0.55-micrometer devices and three-level metal interconnect technology. The processor chip is 1.4 by 1.5 cm in size, contains 1.3 million transistors, and is packaged in a 540-pin ceramic PGA. IEEE 1149.1 JTAG-compliant boundary scan protocol is included for chip test and fault isolation. Fig. 3 is a photomicrograph of the PA 7200 CPU chip.

Instruction Execution

A key feature of the PA 7100 that is retained in the PA 7200 is an execution pipeline highly balanced for both high-frequency operation and very few (compared to most current microprocessors) pipeline stall cycles resulting from data, control, and fetch dependencies.[12] The only common pipeline stall penalties are a one-cycle load-use interlock for any cache hit, a one-cycle penalty for the immediate use of a floating-point result, a zero-to-one-cycle penalty for a mispredicted branch, and a one-cycle penalty for store-load combinations. The PA 7200 improves on the PA 7100 pipeline by removing the penalty for store-store combinations.

Fig. 3. *PA 7200 CPU chip.*



This was achieved by careful timing of off-chip SRAMs, which are cycled at the full processor frequency. Removal of the store-store penalty is particularly helpful for code that has bursts of register stores, such as the code typically found at procedure calls and state saves.

The PA 7200 features an integer superscalar implementation geared to high-frequency operation similar to the PA 7100LC processor.[3] In a superscalar processor, more than one instruction can be executed in a single clock cycle. When two instructions are executed each cycle, this is also referred to as bundling or dual-issuing. In previous PA 7100 processors, only a floating-point operation could be paired with an integer operation. The PA 7200 adds the ability to execute two integer operations per cycle. This will benefit many applications that do not have intensive floating-point operations. To support this integer superscalar capability, the PA 7200 adds a second integer ALU, two extra read ports and one extra write port in the general register stack, a new predecoding block, a new instruction bus, additional register bypassing circuits, and associated control logic.

Instructions are classified into three groups: integer operations, loads and stores, and floating-point operations. The PA 7200 can execute a pair of instructions in a single cycle if they are from different groups or if they are both from the integer operation group. Branches are a special case of integer operations; they can execute with the preceding instruction but not with the succeeding instruction. Double-word alignment is not required for instructions executing in the same cycle. As in the PA 7100, only floating-point operations can bundle across a cache line or page boundaries. The PA 7200 can also execute two instructions writing to the same target register in a single cycle.

The PA 7200 contains three instruction buses that connect the instruction cache interface to two integer ALUs and a floating-point unit. As in the PA 7100, an on-chip double-word instruction buffer assists the bundling of two instructions that may not be double-word aligned. On every cycle, one or two instructions can come from any of four sources (even or odd instructions from the cache, or even or odd instructions from the on-chip buffer) and can go to any of the three destination buses.

The process by which multiple instructions are dispatched to different instruction buses leading to corresponding execution units is called *steering*. The PA 7200 has a very aggressive timing budget for steering and instruction decoding (done in less than one processor cycle); therefore, the steering logic must be fast. In addition, on every cycle, the control logic needs to track which one or two of the three instruction buses contain valid instructions as well as the order of concurrently issued instructions. To avoid having superscalar steering and execution decode logic degrade the CPU frequency, six predecode bits are allocated in the instruction cache for each double word. Data dependencies and resource conflicts are checked and encoded in predecode bits as instructions are moved from memory into the cache, when timing is more relaxed. These six predecode bits are carefully designed so that they are optimal for both the steering circuits and the control logic for proper pipelined execution. Thanks to the optimized design and implementation of these predecode bits and the associated steering circuits and control logic, this path is not a speed-limiting path for the PA 7200 chip and does not obstruct its high-frequency operation.

To minimize area, shift-merge and test condition units are not duplicated in the second ALU. Thus shifts, extracts, deposits, and instructions using the test condition block are limited to one per cycle. Also, instructions with test conditions cannot be bundled with integer operations or loads or stores as their successors. A modern compiler can minimize the effect of these few superscalar restrictions through code scheduling, thereby allowing the processor to exploit much of the instruction-level parallelism available in application code to achieve a low average CPI (cycles per instruction).

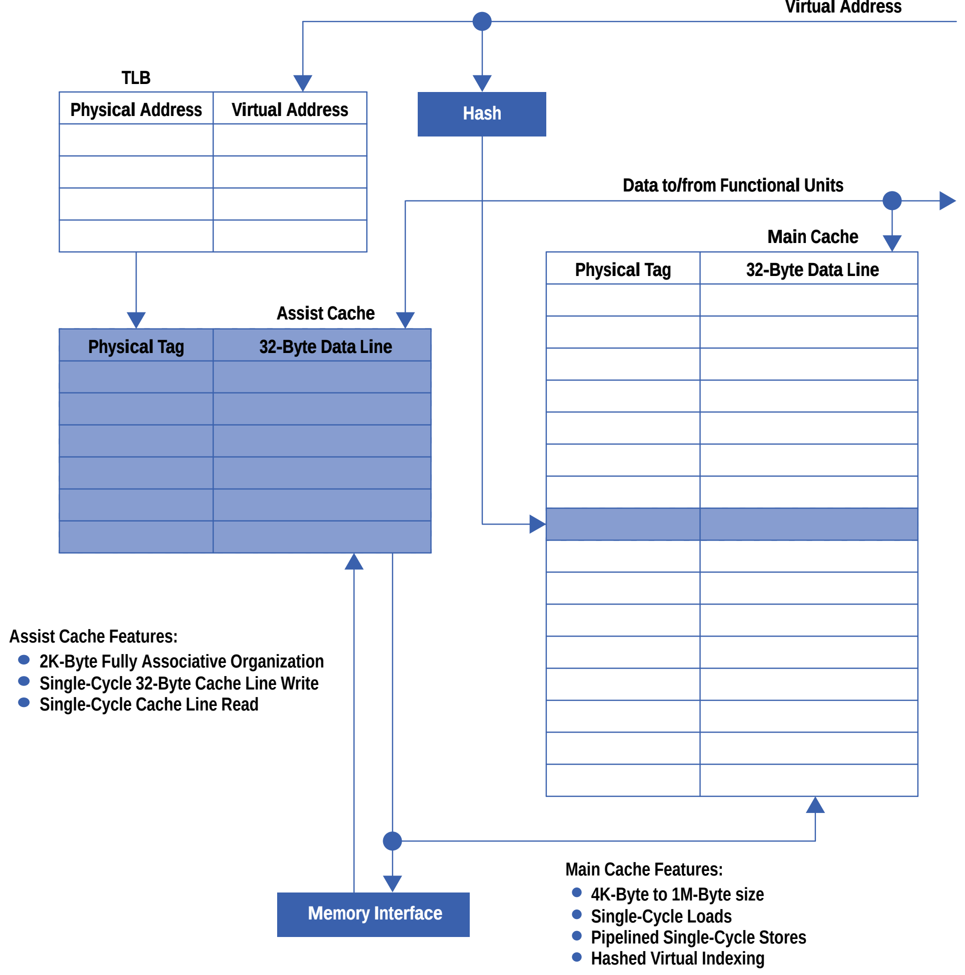
Data Cache Organization

Fig. 4 shows the PA 7200's data cache organization. The chip contains an interface to up to 1M bytes of off-chip direct mapped data cache consisting of industry-standard SRAMs. The off-chip cache is cycled at the full processor frequency and has a one-cycle latency.

The chip also includes a small fully associative on-chip assist cache. Two pipeline stages are associated with address generation, translation, and cache access for both caches, which results in a maximum of a one-cycle load-use penalty for a hit in either cache. The on-chip assist cache combined with the off-chip cache together form a level-1 cache. Because this level-1 cache is accessed in one processor cycle and supports a large cache size, no level-2 cache is supported. The ability to access the large off-chip cache with low latency greatly reduces the CPI component associated with cache-resident memory references. This is particularly helpful for code with large working data sets.

The on-chip assist cache consists of 64 fully associative 32-byte cache lines. A content-addressable memory (CAM) is used to match a translated real line address with each entry's tag. For each cache access, 65 entries are checked for a valid match: 64 assist cache entries and one off-chip cache entry. If either cache hits, the data is returned directly to the appropriate functional unit with the same latency. Aggressive self-timed logic is employed to achieve the timing requirements of the assist cache lookup.

Fig. 4. PA 7200 data cache organization.



Lines requested from memory as a result of either cache misses or prefetches are initially moved to the assist cache. Lines are moved out of the assist cache in first-in, first-out order. Moving lines into the assist cache before moving them into the off-chip cache eliminates the thrashing behavior typically associated with direct mapped caches. For example, in the vector calculation:

```
for i:  =  0 to N do
A[i] :  =  B[i] + C[i] + D[i]
```

if elements A[i], B[i], C[i], and D[i] map to the same cache index, then a direct mapped cache alone would thrash on each element of the calculation. This would result in 32 cache misses for eight iterations of this loop. With an assist cache, however, each line is moved into the cache system without displacing the others. Assuming sequential 32-bit data elements, eight iterations of the loop causes only the initial four cache misses.

Larger caches do not reduce this type of cache thrashing. While modern compilers are often able to realign data structures to reduce or eliminate thrashing, sufficient compile time information is not always available in an application to make the correct optimization possible. The PA 7200's assist cache eliminates cache thrashing extremely well with minimal hardware and without compiler optimizations.

Lines that are moved out of the assist cache can conditionally bypass the off-chip cache and move directly back to memory. A newly defined *spatial locality only* hint can be specified in load and store instructions to indicate that data exhibits spatial locality but not temporal locality. A data line fetched from memory for an instruction containing the spatial locality hint is moved into the assist cache like all other lines. Upon replacement, however, the line is flushed back to memory instead of being moved to the off-chip cache. This mechanism allows large amounts of data to be processed without polluting the off-chip cache. Additionally, cycles are saved by avoiding one or two movements of the cache line across the 64-bit interface to the off-chip cache.

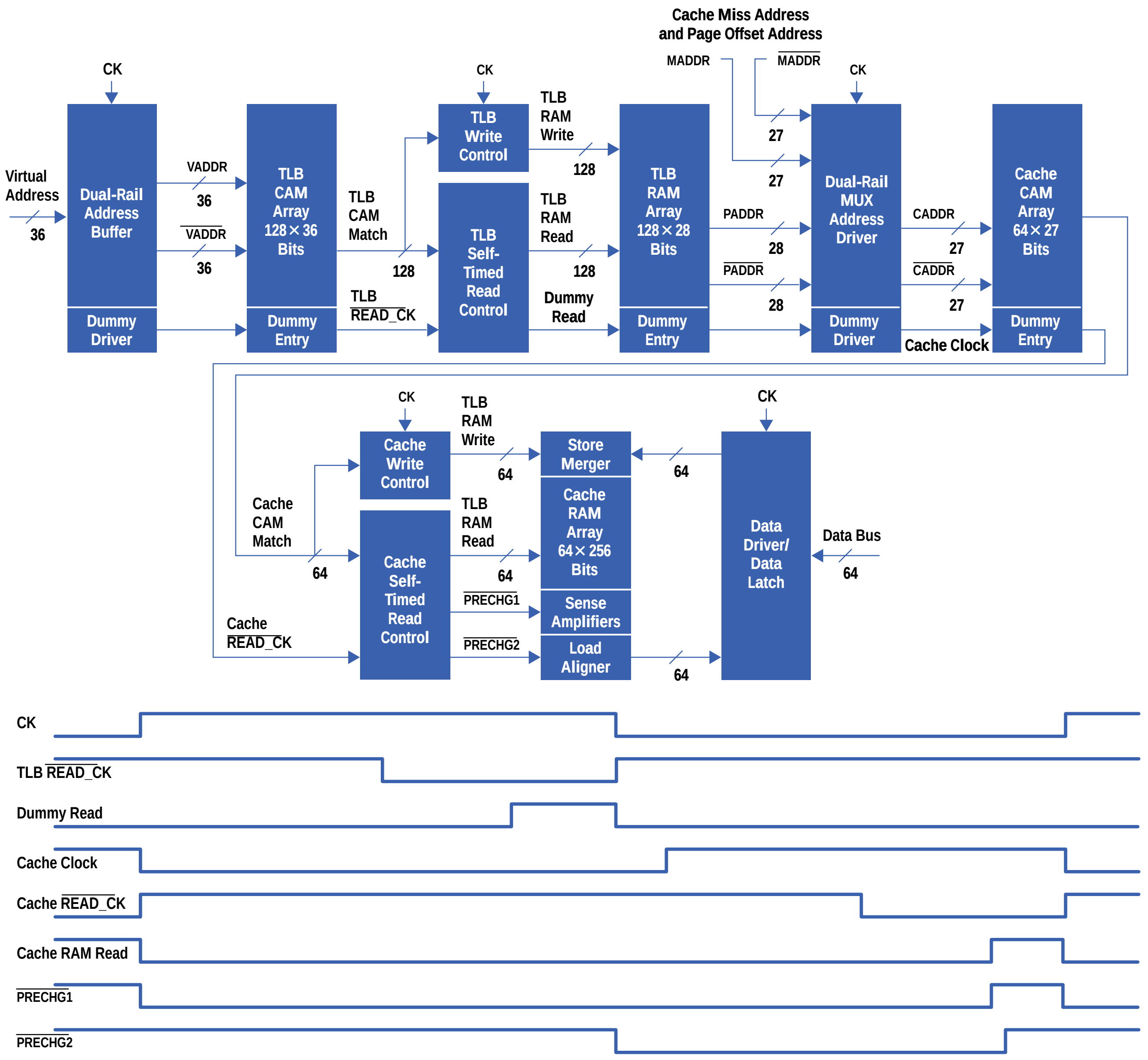
The assist cache allows prefetches to be moved into the cache system in a single cycle. Prefetch returns are accumulated independently of pipeline execution. When the complete line is available, one data cache cycle is used to insert the line into the on-chip assist cache. If an instruction that is not using the cache is executing, no pipeline stalls are incurred.

Because the assist cache is accessed using a translated physical address, it adds an inherently critical speed path to the chip microarchitecture. An assist cache access consists of virtual cache address generation, translation lookaside buffer (TLB) lookup to translate the virtual address into a physical address, and finally the assist cache lookup. The TLB lookup and assist cache lookup need to be completed in one processor cycle or 8.3 ns for 120-MHz operation. To meet the speed requirements of this path a combination of dynamic and self-timed circuit techniques is used.

The TLB and assist cache are composed of content-addressable memory (CAM) structures, which differ from more typical random-access memory (RAM) structures in that they are accessed with data, which is matched with data stored in the memory, rather than by an index or address. A typical RAM structure can be broken into two halves: an address decoder and a memory array. The input address is decoded to determine which memory element to access. Similarly, a CAM has two parts: a match portion and a memory array. In the case of the assist cache, the match portion consists of 27-bit comparators that compare the stored cache line tag with the translated physical address of the load or store instruction. When a match is detected by one of the comparators, then that comparator dumps the associated cache line data.

Fig. 5 shows the timing of an access to the TLB and assist cache.

Fig. 5. *PA 7200 TLB and assist cache timing.*


Cache Miss Address and Page Offset Address
MADDR
$\overline{\text{MADDR}}$
CK
Virtual Address
36
Dual-Rail Address Buffer
Dummy Driver
VADDR
36
$\overline{\text{VADDR}}$
36
TLB CAM Array 128 × 36 Bits
Dummy Entry
TLB CAM Match
128
TLB $\overline{\text{READ\_CK}}$
TLB Write Control
TLB Self-Timed Read Control
TLB RAM Write
128
TLB RAM Read
128
Dummy Read
TLB RAM Array 128 × 28 Bits
Dummy Entry
PADDR
28
$\overline{\text{PADDR}}$
28
27
27
Dual-Rail MUX Address Driver
Dummy Driver
CADDR
27
$\overline{\text{CADDR}}$
27
Cache Clock
Cache CAM Array 64 × 27 Bits
Dummy Entry
Cache CAM Match
64
Cache $\overline{\text{READ\_CK}}$
Cache Write Control
Cache Self-Timed Read Control
TLB RAM Write
64
TLB RAM Read
64
$\overline{\text{PRECHG1}}$
$\overline{\text{PRECHG2}}$
Store Merger
Cache RAM Array 64 × 256 Bits
Sense Amplifiers
Load Aligner
64
64
Data Driver/ Data Latch
Data Bus
64
CK
TLB $\overline{\text{READ\_CK}}$
Dummy Read
Cache Clock
Cache $\overline{\text{READ\_CK}}$
Cache RAM Read
$\overline{\text{PRECHG1}}$
$\overline{\text{PRECHG2}}$

This single 8.3-ns clock cycle path is broken into multiple subsections using self-timed circuits. An access begins when the single-ended virtual address is latched and converted to complementary predischarged values VADDR and $\overline{\text{VADDR}}$ in the TLB address buffer on the rising edge of CK. These dual-rail signals are then used to access the CAM array. A dummy CAM array access, representing the worst-case timing through the CAM array, is used to initiate the TLB RAM access. If any of the CAM entries matches the $\overline{\text{VADDR}}$, then the completion of the dummy CAM access, represented by TLB$\overline{\text{READ_CK}}$, enables the TLB read control circuits to drive one of the TLB RAM read lines. The precharged RAM array is then read and a differential predischarged physical address is driven to the assist cache. A similar access is then made to the assist cache CAM and RAM structures to produce data on the rising edge of CK. A precharged load aligner is used to select the appropriate part of the 256-bit cache line to drive onto the data bus and to perform byte swapping for big-to-little-endian data format conversion. Although this path contains tight timing budgets, careful circuit design and physical layout ensure that it does not limit the processor frequency.

The basic structure of the external cache remains unchanged from the PA 7100 CPU. Separate instruction (I) and data (D) caches are employed, each connected to the CPU by a 64-bit bidirectional bus. The cache is virtually indexed and physically tagged to minimize access latency. The I-cache data and tag are addressed over a common address bus, IADH. The D-cache data has a separate address bus, DADH, and the D-cache tag has a separate address bus, TADH. Used in conjunction with an internal store buffer for write data, the split D-cache address allows higher-bandwidth stores to the D-cache. Instead of a serial read-modify-write, stores can be pipelined so that TADH can be employed for the tag read of a new store instruction while DADH is used to write the data from the previous store instruction.

As in the PA 7100 CPU, the PA 7200 CPU cache interface is tuned to work with asynchronous SRAMs by creating special clock signals for optimal read and write timing. The cache is read with a special latch edge that allows *wave pipelining*, that is, a second read is launched before the first read is actually completed. The cache is written using two special clocks that manipulate the write enable and output enable SRAM controls for a minimum total write cycle time.

The design team worked closely with several key SRAM vendors to develop a specification for a 6-ns SRAM with enhanced write speed capabilities. These new SRAMs allow both of the caches to operate at the CPU clock frequency. The CPU can be shipped with equal-sized instruction and data caches of up to 1M bytes each. As in the PA 7100 CPU, a read can be finished in one clock cycle. However, to match the bandwidth of the Runway bus and to increase the performance of store-intensive applications, a significant timing change was made to improve the bandwidth for writes to the cache. The PA 7200 CPU achieves a quasi-single-cycle write: a series of N writes requires N+1 cycles. The one-cycle overhead is required for turning the bus around from read to write, that is, one cycle is required to turn off the SRAM drivers and allow the CPU drivers to take over. No penalty is incurred in transitioning from write to read.

Prefetching Mechanisms

A significant amount of execution time is spent waiting for data or instructions to be returned from memory. In an HP 9000 K-class system running transaction processing applications, an average of about one cycle per instruction can be attributed to the processor waiting for memory. The total CPI for such an application is about 2. Execution time can therefore be greatly reduced by reducing the number of cycles the processor spends waiting for memory. The PA 7200 incorporates hardware and software prefetching mechanisms, which initiate memory requests before the data or instructions are used.

Instruction Prefetching. The PA 7200 implements an efficient instruction prefetch algorithm. Instruction fetch requests are issued speculatively ahead of the instruction execution stream. Multiple instruction prefetch requests can be in flight to the memory system simultaneously. Issuing multiple prefetches ahead of the execution stream works well when linear code segments are initially encountered. This instruction prefetching scheme yields a 9% performance speedup on transaction processing benchmarks.

Data Prefetching. The PA-RISC instruction set includes a class of instructions that modify the base value in a general register by an immediate displacement or general register index value. An example is LDWX,m r1(r2),r3. The LDWX (load word indexed) instruction with a modify completer (,m) loads the value at the address contained in register r2 into register r3, and then adds r1 to r2 (i.e., load r2 –> r3; r1 + r2 –> r2). The PA 7200 can use this class of instructions to speculate what data may soon be accessed by the code stream. If the load r2 in the above example is a cache miss, a prefetch is issued to the address calculated by the base register modification (r1 + r2). The PA 7200 uses this base register modification to speculate where a future data reference will occur. For example, if r1 contains line 0x40 and r2 contains line 0x100 and no lines are initially in the cache, then this instruction initiates a request for line 0x100 in response to the cache miss and line 0x140 is prefetched. If the line 0x140 is later used, some or all of the cache miss penalty is avoided.

When a line is prefetched, it is moved into the assist cache and tagged as being a prefetched line. When a prefetched line is later referenced by the code stream, another prefetch is launched. Continuing with the above example, if this load instruction were contained in a loop, on the first iteration of the loop lines 0x100 and 0x140 would be requested from memory. On the second iteration line 0x140 is referenced. The assist cache detects this as the first reference to a prefetched line and initiates a prefetch of line 0x180. This allows memory requests to stay ahead of the reference stream, reducing the stall cycles associated with memory latency.

The PA 7200 allows four data prefetch requests to be outstanding at one time. These prefetches can be used for either prefetches along multiple data reference streams or farther ahead on one data reference stream. Returning to the vector example,

```
for i :  =  0 to N do
A[i] :  =  B[i] + C[i] + D[i]
```

each new cache line entered will cause four new prefetch requests to be issued: one for each vector. On the other hand, if the processor were doing a block copy:

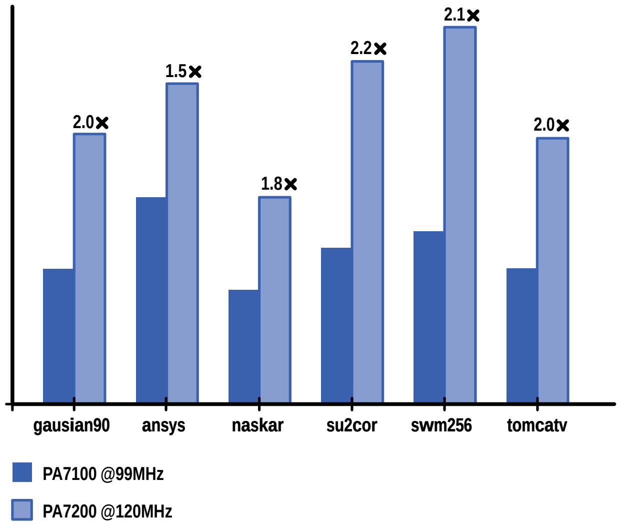
```
for i :  =  0 to N
A[i] :  =   B[i]
```

then it could prefetch two lines ahead of each reference stream.

Reducing Average Memory Access Time

A number of features have been combined in the PA 7200 to minimize the average memory access time (the average number of cycles used for a memory reference).[13] These features together provide excellent performance speedups on a number of applications that stress the memory hierarchy. Fig. 6 compares the performance of the PA 7200 and the PA 7100 on a number of technical benchmarks. To minimize the average memory access time associated with cache hits, the large low-latency off-chip cache from the PA 7100 design has been retained and enhancements made to allow single-cycle stores. The PA 7200 improves on the PA 7100 by reducing cache misses by minimizing compulsory, capacity, and conflict cache misses.

Fig. 6. *A number of features that minimize the average memory access time allow the PA 7200 CPU to outperform its predecessor the PA 7100 on technical benchmarks.*



The PA 7200 reduces conflict misses by adding effective associativity to entries of the main cache. This is done without the overhead required for a large multiset associative cache. Traditionally caches have been characterized as direct mapped, multiset associative, or fully associative. The PA 7200 assist cache effectively adds dynamically adjusted associativity to main cache entries. As miss lines are brought into the assist cache, the entries with the same cache index mapping in the main cache are not immediately replaced. This allows multiple cache lines with the same index to reside in “the cache” at the same time. All assist cache entries can be filled with lines that map to the same off-chip cache index, or they can be filled with entries that map to various indexes. This eliminates the disastrous thrashing that can occur with a direct mapped cache, as discussed earlier.

The PA 7200 reduces compulsory cache misses by prefetching lines that are likely to be used. When the software has the information necessary at compile time to anticipate what data is needed, the base register modification class of load and store instructions can be used to direct prefetching. If no specific direction is added to code or if old code is being run, then base register modifying loads and stores can still be used by the hardware to do effective prefetching. The processor can also be configured to use loads and stores that do not modify base registers to initiate speculative requests. Because memory bandwidth is limited, care was taken to minimize the amount of bad prefetching while maximizing the speedup realized by issuing memory requests speculatively. Both old code traces and new compiler optimizations were investigated to determine the best set of prefetching rules.

In addition to the large caches supported by the PA 7200, capacity misses are reduced by selectively allocating lines to the off-chip cache if they benefit from being moved to the off-chip cache. More effective use can be made of a given cache capacity by only moving data that exhibits temporal locality to the off-chip cache. The assist cache provides an excellent location for use-once data. The spatial locality only (,SL) hint associated with load and store instructions allows code to identify which data is use-once (or simply too large to be effectively cached), thereby reducing capacity misses. The ,SL hint is encoded in previously reserved load and store instruction fields. Large analytic applications and block move and clear routines achieve excellent speedups from this new cache hint.

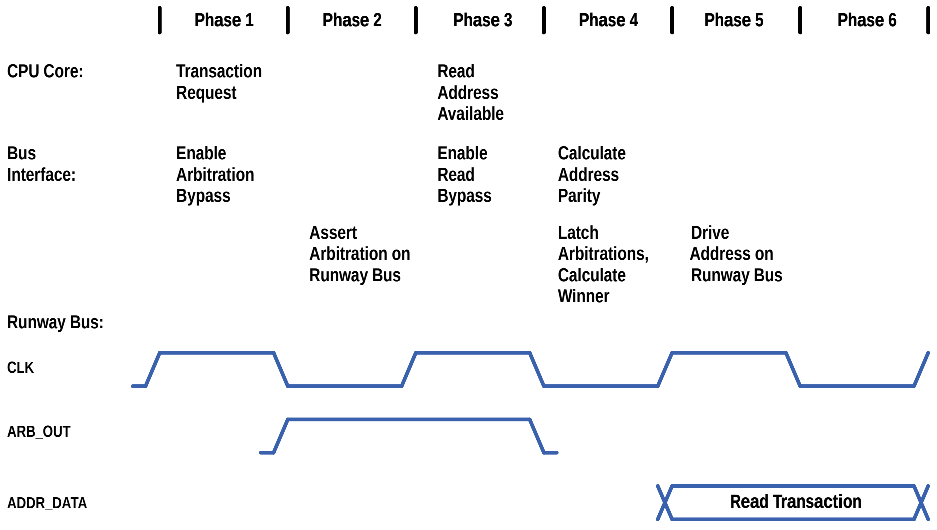
Bus Interface

The PA 7200's Runway bus interface is carefully tuned to the requirements and capabilities of the processor core. The interface has several features that minimize transaction latency, reduce processor cost, and take advantage of particular attributes of the CPU core to simplify interface design. The bus interface contains a cache coherence queue and transaction buffers, arbitration logic, and logic to support multiple processor-to-bus-frequency ratios. The bus interface also implements an efficient *double snoop*† algorithm for coherent transaction management.

The PA 7200 connects directly to the Runway bus without transceivers or interface chips. Without this layer of external logic, system cost is reduced while performance is increased because of lower CPU-to-bus latency. Special system and circuit designs allow the Runway bus to run at a frequency of 120 MHz while maintaining connectivity to six loads. Negative-hold-time receiver design and tight skew control prevent races when drivers and receivers operate from the same clock edge. A read transaction is issued in one bus cycle and the 32-byte memory return is transferred in four cycles, resulting in a peak sustainable bandwidth of 768 megabytes per second. To take advantage of the high bus bandwidth, the PA 7200 can have up to six memory reads in flight at the same time.

To minimize read transaction latency, the PA 7200 asserts and captures arbitration signals on the half cycle (phase), as shown in Fig. 7. The processor core communicates its intent to initiate a transaction in the first phase, allowing the interface to assert its bus arbitration signal on the second phase.

Fig. 7. *With numerous bypass paths, latency between the CPU core and the Runway bus is minimized. As soon as the CPU detects a cache miss, the bus interface arbitrates for the system bus in half a cycle. As soon as the cache miss address is available, it is routed to the interface in half a cycle, where its bus parity is generated in another half-cycle. Performance is maximized in the common case of little bus traffic, when the CPU wins bus arbitration immediately.*



The transaction address information, only available on the third phase, is then forwarded from the processor core to the bus interface. In the common case where there is no contention for the Runway bus, the address is driven onto the bus in the next cycle. Read and write buffers, included in the bus interface to decouple the CPU core in case arbitration is not immediately won, are bypassed in the common case to reduce latency.

Transactions from the read and write buffers are issued by the bus interface with fixed priorities. Snoop data has the highest priority, followed by read requests, then the write of cache victims. When the memory controller cannot handle new read requests and the read and write buffers are full, the bus interface will issue the write transaction before the read to make best use of the bus bandwidth available.

Since transactions on the Runway bus are always accepted (and never rejected or retried at the expense of bus bandwidth), each processor acting as a third party must be able to accept a burst of coherent transactions. Since there are times when the CPU core is busy and cannot accept a snoop, the bus interface implements a ten-transaction-deep queue for cache snoops and a three-transaction-deep queue for TLB snoops. With deep coherency queues, a large number of coherent transactions from several processors can be outstanding without the need to invoke flow control.

Processor-to-bus frequency ratios of 1:1, 3:2, and 4:3 are provided for higher-frequency processor upgrades. Using a ratio algorithm that requires the bus clock to be synchronous with the processor clock ensures that the ratio logic does not

† A snoop, also known as a cache coherency check, is the action performed by all processors and I/O adapters when they observe a coherent transaction issued by another module. Each module performing the snoop must check its cache for the address of the current transaction, and if found, change the state of that cache address. Cache state transitions are described in *Article 2.*

impart synchronization delays typical of systems with asynchronous clock domains. For any ratio, the worst-case delay is less than one CPU clock cycle, and in the best case, data transmission does not incur any delay.

To minimize processor pipeline stalls resulting from multiprocessor interference, transactions at the head of the coherency queue are forwarded to the CPU core in two steps. First, the core is sent a lightweight query, which steals one cycle of cache bandwidth. A low-latency response is received from the off-chip and assist caches. Only when a cache state modification is required is a second full-service query forwarded to the CPU core. Since the vast majority of cache snoops result in misses, this double snoop approach allows the PA 7200 to achieve higher multi– processor performance without the added cost and complexity of a dual-ported cache or duplicate cache tags.[14]

PA 7200 Circuit Translation

Most of the PA 7200 circuit designs, artwork, and physical design methodology are based upon and leveraged from the PA 7100 CPU, which was designed using HP's CMOS26 IC process, tools, and libraries. However, aggressive performance and cost goals required that the PA 7200 be fabricated using the faster, denser CMOS14 IC process also under development. To completely redesign and lay out existing PA 7100 circuits for the CMOS14 process would have been an inefficient use of resources and would have greatly extended the design phase. Therefore, the entire PA 7200 was designed using the existing CMOS26 technology, and the artwork was then automatically translated to and reverified in the CMOS14 process.

Unfortunately, automatic translation faced two global issues. First, CMOS26 is a 5.0V (nominal) process but CMOS14 was originally specified for 4.0V operation. Simulations showed that the speed of a few common circuit topologies did not scale linearly into the target technology because of the lower supply voltage. Detailed investigation by the CMOS14 development group concluded that raising the supply voltage by 10% was feasible and the process was fully qualified for operation at 4.4V. This was sufficient for these circuits to meet the speed improvement goal.

Secondly, CMOS26 layout rules do not scale uniformly into the respective rules for CMOS14, since each component of a process technology has different physical and manufacturing constraints. A simple gate-shrink algorithm, which only reduces FET effective gate length, could have provided a 20% transistor speed improvement. Without overall area reduction, the extra PA 7200 functionality dictates a die size much larger than the PA 7100 and this approach would result in slower wire speeds and a sharp increase in manufacturing cost. With aggressive scaling, a more complex translation algorithm, and a limited number of engineering adjustments to the layout and electrical rules, the CMOS14 version achieves a 20% overall speed improvement along with a 38% power reduction from the original CMOS26 design.

Translation Methodology. The methodology that was developed accommodates CMOS26 designs and translated CMOS14 artwork in parallel, is generally transparent, and merges smoothly with the existing design environment. A hierarchical (block-level) translation methodology was chosen because it provides many advantages over the more traditional flat (mask-level) translation. Important reasons for selecting this approach were:

- *Algorithm flexibility*. The optimal translation algorithm is not required to guarantee that every pathological CMOS26 layout, and more important, all existing PA 7100 blocks are translated to a legal CMOS14 layout as long as a manageable number of violations result and are easily correctable by hand. Hierarchical methods imply editing only unique instances of a violation at the block level, rather than the entire set on a flattened mask.
- *Design modularity*. Having parallel hierarchies containing both CMOS26 and CMOS14 blocks enables additional flexibility. Translated artwork can be read directly by the front-end editors for electrical simulation and other purposes. On the top-level routing blocks, CMOS14 layouts using a tighter metal pitch were a necessary alternative to the translated CMOS26 versions.
- *Concurrent methodology*. Translated artwork is available for mask generation along with the original block. Flat translation is serialized and for complex algorithms implies a costly delay after each design release. Moreover, having a complete, hierarchical CMOS14 artwork database allowed subsequent chip revisions to be released using incremental changes made directly to the CMOS14 artwork.

Many operations in the translation algorithm are complicated by hierarchical junctions (these would disappear with a flat translation.) A hierarchical junction is any connection between objects in separate blocks. If individual artwork features touching or extending beyond hierarchical boundaries are further shrunk by a fixed distance after being reduced by the scaling coefficient, gaps will occur at the parent junctions that cannot always be filled automatically. A subtle but more troublesome scaling problem is caused by snapping the location of child instances to the grid resolution, which creates shape misalignments or gaps at parent-child or child-child junctions if origins round in a different direction. This effect can be cumulative, and becomes significant for junctions that span multiple hierarchical levels. Increased database size and consistency checking are other drawbacks of a block-oriented translation.

A final check was added after CMOS14 layout verification to hierarchically compare ports, signals, and connectivity between the CMOS26 and CMOS14 artwork netlists. This was necessary since hand corrections made to the translated CMOS14 layout could introduce new design errors.

Translation Algorithm. Any scaling coefficient should ensure that all minimum widths, spaces, and exact-size shapes from CMOS26 be translated to CMOS14 such that each edge pair snaps to the grid resolution (0.05-µm) in the same direction. There are several natural solutions to ensure that 1.0-µm (drawn) minimum features in CMOS26 always become 0.6-µm minimum features in CMOS14. For example:

- Scale by $\alpha = 0.8$ and then further shrink interconnect by 0.2 μm.
- First shrink interconnect by 0.2 μm and then scale by $\alpha = 0.75$.

The second option is only practical for library blocks since it is too aggressive for interconnect with minimum contacted pitch and provides less margin for the effects of uneven grid snapping. The detailed algorithm is based upon the first option, with additional manipulations of n-well regions, FET gate extensions, contact sizes, interconnect contact enclosure, and interlayer contact spacing. These operations have parasitic effects which can create notches and narrow corners and are usually correctable by automatically filling new width and spacing violations.

There were still a residual number of geometrical cases that could not be fully translated by any reasonable tool or heuristic. In these cases we either waived the layout rules where margin was available or made extra efforts to repair rule violations by hand. Although many of these violations did occur, the vast majority resulted either from the hierarchical phenomena described earlier or from fundamental scaling issues with certain contact structures and latch-up prevention rules. In no case was any significant block relayout required, however.

Scaling-Sensitive Circuits. Although algorithmic translation of PA 7200 circuits generally improves electrical performance and decreases parasitic effects, there are a few exceptional circuits with different characteristics. In general, these were abnormally sensitive to transistor sizing ratios, noise caused by coupling, voltage shifts caused by charge sharing, small variations in processing parameters, or the reduced 4.4V high level. Additionally, total resistance in the third layer of metal can increase after translation and cause routing delays to improve less than the basic scaling assumptions predict.

Summary

The design goal for the PA 7200 was to increase the performance of Hewlett-Packard computer systems on real-world applications in a variety of markets while maintaining a high degree of price/performance scalability and a low system component count. General application performance is improved through an increase in operating frequency, a second integer ALU for enhanced superscalar execution, and improved store instruction performance. For applications that operate on large data sets, such as typical analytic and scientific applications, the hardware prefetching algorithms and fully associative assist cache implemented in the PA 7200 provide excellent performance increases. In addition, the processor includes a high-bandwidth, low-latency multiprocessor bus interface to support cost-effective, high-performance, one-way to four-way multiprocessor systems, which are ideal for technical or commercial platforms, without additional interface chips. Additionally, the PA 7200 is scalable from desktop workstations to many-way multiprocessor corporate computing platforms and supercomputers.

Acknowledgments

The authors would like to thank all the different teams who contributed to the successful design of the PA 7200 chip. The design and implementation was primarily done by many individuals from the Computer Technology Laboratory and the Cupertino Systems Laboratory in Cupertino, California and several individuals from the Engineering Systems Laboratory in Fort Collins, Colorado. Many thanks also to several teams whose work was responsible for many key design decisions made. This includes the Integrated Circuits Business Division in Fort Collins and the Systems Performance Laboratory and the California Languages Laboratory in Cupertino.

References

1. M.J. Mahon, et al, "Hewlett-Packard Precision Architecture: The Processor," *Hewlett-Packard Journal*, Vol. 37, no. 8, August 1986, pp. 4-21.
2. R.B. Lee, "Precision Architecture," *IEEE Computer*, Vol. 22, no. 1, January 1989, pp.79-91.
3. P. Knebel, et al, "HP's PA 7100LC: A Low-Cost Superscalar PA-RISC Processor," *Compcon Digest of Papers*, February 1993, pp. 441-447.
4. E. Delano, et al, "A High-Speed Superscalar PA-RISC Processor," *Compcon Digest of Papers*, February 1992, pp. 116-121.
5. M. Forsyth, et al, "CMOS PA-RISC Processor for a New Family of Workstations," *Compcon Digest of Papers*, February 1991, pp. 202-207.
6. D. Tanksalvala, et al, "A 90-MHZ CMOS RISC CPU Designed for Sustained Performance," *ISSCC Digest of Technical Papers*, February 1990, pp. 52-53.
7. B.D. Boschma, et al, A 30-MIPS VLSI CPU," *ISSCC Digest of Technical Papers*, February 1989, pp. 82-83.
8. J. Yetter, et al, "A 15-MIPS 32b Microprocessor," *ISSCC Digest of Technical Papers*, February 1987, pp. 26-27.
9. D. Fotland, et al, "Hardware Design of the First HP Precision Architecture Computers," *Hewlett-Packard Journal*, Vol. 38, no. 3, March 1987, pp. 4-17.
10. G. Kurpanek, et al, "PA 7200: A PA-RISC Processor with Integrated High-Performance MP Bus Interface," *Compcon Digest of Papers*, February 1994, pp. 375-382.
11. E. Rashid, et al, "A CMOS RISC CPU with on-chip Parallel Cache," *ISSCC Digest of Technical Papers*, February 1994.

12. T Asprey, et al, "Performance Features of the PA 7100 Microprocessor," *IEEE Micro*, June 1993, pp. 22-35.
13. J. Hennessy and D. Patterson, *Computer Architecture, A Quantitative Approach*, Morgan Kaufmann Publishers, 1990.
14. K. Chan, et al, "Multiprocessor Features of the HP Corporate Business Servers," *Compcon Digest of Papers*, February 1993, pp. 330-337.

▶ Go to *Article 4*
▶ Go to Table of Contents
▶ Go to HP Journal Home Page