

# M I C R O P R O C E S S O R

www.MPRonline.com



THE INSIDER'S GUIDE TO MICROPROCESSOR HARDWARE



## MIPS THREADS THE NEEDLE

*MIPS32 34K: The First Licensable Multithreaded Processor Core*

*By Tom R. Halfhill {2/27/06-01}*

Microprocessor architects have explored many paths to high performance, including high clock frequencies, superscalar pipelines, application-specific extensions, very long instruction words (VLIW), and multicore processors. All those techniques and

more are available in embedded-processor cores licensed as synthesizable intellectual property (IP). Now MIPS Technologies is adding another option: the first licensable processor cores with hardware-enabled simultaneous multithreading.

The new MIPS32 34K family consists of four 32-bit processor cores, all related to the MIPS32 24KE family introduced at Spring Processor Forum 2005. (See *MPR 5/31/05-01*, "White Paper: The MIPS32 24KE Core Family.") The key difference between the 24KE and 34K families is pipelined multithreading. Instructions from as many as five different tasks can pass through the nine-stage pipeline of a 34K processor at the same time.

Although the uniscalar 34K can issue only one result per clock cycle, multithreading allows it to reduce the overhead of context switching and hide the latency of slow operations, such as memory-dependent loads and stores. This technique is known as simultaneous multithreading (SMT) or thread-level parallelism. Note that for the purposes of this discussion, a thread can be *any* executable process—even an operating system—not just a microthread within a program.

### **Multithreading Reaches a Watershed**

MIPS isn't the first company to use SMT, which dates to the early 1990s. (See *MPR 7/14/97-03*, "Multithreading Comes of Age.") In 1999, DEC announced that Alpha 21464 server processors would use SMT, but Alpha's unpopularity and premature demise did little to promote the technology. (See

*MPR 12/6/99-01*, "Compaq Chooses SMT for Alpha.") In 2001, Intel brought SMT to the mainstream by announcing Hyper-Threading, a limited two-way version of thread-level parallelism that has appeared in several Pentium 4 and Xeon processors for PCs and servers. (See *MPR 9/17/01-01*, "Intel Embraces Multithreading.")

More recently, embedded-processor architects have seized upon SMT as a power-efficient path to higher performance. That's the big attraction for MIPS. Multithreading avoids the pipeline duplication of superscalar execution and the core duplication of multicore processing, although it may be combined with those and other techniques. In addition, thread-level parallelism is easy to exploit in many kinds of embedded software. Those advantages led a startup, Ubicom, to introduce a tiny multithreaded packet processor in 2003. (See *MPR 4/21/03-01*, "Ubicom's New NPU Stays Small.")

In 2005, the technology reached a watershed when Intel, Raza Microelectronics (RMI), and Sun Microsystems introduced powerful processors that combine multithreading with multiple cores, which is called chip multithreading (CMT). (See *MPR 5/9/05-01*, "A Day At the Races," *MPR 5/17/05-01*, "A New MIPS Powerhouse Arrives," and *MPR 1/3/06-01*, "Sun's Niagara Begins CMT Flood.")

Not until now, however, have ASIC and SoC developers been able to license a multithreaded embedded-processor core for their own chip designs. The MIPS32 34K family fulfills a promise made at Microprocessor Forum 2003,



when MIPS announced the multithreading application-specific extension (MT ASE) for future MIPS processors. (See *MPR 11/10/03-01*, “Multithread Technologies Disclosed at MPE.”) MIPS says multithreading can dramatically improve performance while preserving compatibility with existing single-threaded software.

MIPS has already licensed the 34K to five customers, two of which prefer to remain anonymous for now. The public customers are iVivity, Mobileye, and PMC-Sierra. Georgia-based iVivity makes storage processors. Mobileye, based in the Netherlands, is integrating a 34Kf core into its future EyeQ-2 chip, an embedded controller for automotive collision-avoidance systems. PMC-Sierra, based in Silicon Valley, is a longtime MIPS licensee that will most likely use the 34K in high-performance network and communications processors.

### 34K Family Resembles 24KE

MIPS is introducing four members of the MIPS32 34K family, consistent with previous variations within the 24KE, 24K, and 4KE families of 32-bit embedded-processor cores. As Table 1 shows, all four new processors share the same basic CPU core, with DSP extensions inherited from the 24KE family. Variations within the 34K family include an optional FPU plus the option of adding user-defined extensions—a Pro Series feature that MIPS calls CorExtend. (See *MPR 3/3/03-01*, “MIPS Embraces Configurable Technology.”) Both features affect the core’s size and power consumption.

Feature	MIPS32 34Kc	MIPS32 34Kf	MIPS32 34Kc Pro	MIPS32 34Kf Pro
Architecture	MIPS32 R2	MIPS32 R2	MIPS32 R2	MIPS32 R2
CPU Core	34K	34K	34K	34K
Synthesizable	Yes	Yes	Yes	Yes
Pipeline Depth	9 stages	9 stages	9 stages	9 stages
MIPS16e	Yes	Yes	Yes	Yes
I-Cache	0–64K	0–64K	0–64K	0–64K
D-Cache	0–64K	0–64K	0–64K	0–64K
MMU	Yes	Yes	Yes	Yes
Integer Mul-Div	32x32-bit	32x32-bit	32x32-bit	32x32-bit
Bus Interface	OCB 2.x 64-bit	OCB 2.x 64-bit	OCB 2.x 64-bit	OCB 2.x 64-bit
Coprocessor I/F	64-bit COP2	64-bit COP2	64-bit COP2	64-bit COP2
FPU (32/64-Bit)	—	Yes	—	Yes
DSP ASE	Yes	Yes	Yes	Yes
CorExtend	—	—	Yes	Yes
Core Freq (90nm)	500MHz	500MHz	500MHz	500MHz
Core Size (Base)	2.1mm <sup>2</sup>	n/a	2.1mm <sup>2</sup>	n/a
Power @ 500MHz	280mW	n/a	280mW	n/a
(Base Core)	1.0V		1.0V	
Availability	Now	Now	Now	Now

**Table 1.** Members of the new MIPS32 34K family have most features in common, differing only by whether individual members have an FPU and/or CorExtend. Clock frequencies and power-consumption estimates in this table assume a core supply of 1.0V and a generic 90nm CMOS fabrication process under worst-case conditions. The estimated core size—extracted from a full-layout GDSII database—excludes caches and assumes a configuration supporting four simultaneous threads. Core-size and power-consumption estimates for the FPU-equipped processors are not available (n/a) but will be slightly greater than for the 34Kc.

The simplest MIPS32 34K core is the 34Kc, which has neither an FPU nor CorExtend. Next comes the 34Kf, which has an FPU, again without CorExtend. The 34Kc Pro has CorExtend but no FPU. The highest-end core is the 34Kf Pro, which has both an FPU and CorExtend. Thanks to the MIPS architecture’s workstation/server heritage, the optional FPU available for these cores is one of the best available for a licensable embedded processor. It supports single- and double-precision floating-point math and complies with the IEEE 754 standard. CorExtend is a powerful option, too, although the configurable-processor technology from competitors ARC International and Tensilica is superior.

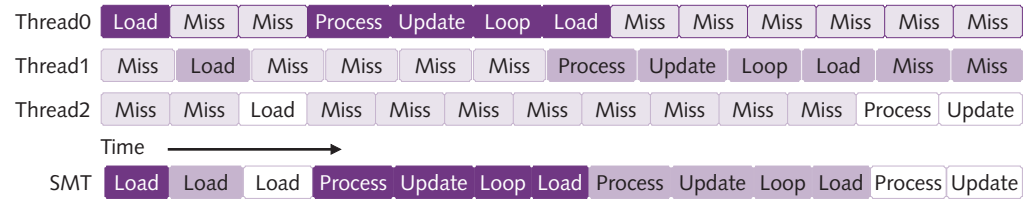
SMT is the feature that distinguishes the 34K from all other licensable embedded-processor cores. MIPS took pains to add this capability without disrupting the well-established MIPS architecture or exceeding the power envelope expected of an embedded processor. Perhaps the most critical decision was the number of threads to support. In theory, the number of simultaneous threads is limited only by the depth of the pipeline, because at any moment, each pipe stage can be processing an instruction from a different thread. However, the maximum number of threads isn’t necessarily the optimal number, especially for an embedded processor. Each thread requires a duplicate register file, program counter, and other structures to store its context while the processor switches from one thread to another. This overhead inflates the processor’s gate count and power consumption.

At design time, customers may configure the 34K to support two to five simultaneous threads. (Although a single-threaded implementation is possible, the 24KE is a better choice for applications needing only one thread.) The maximum of five threads seems odd, because it requires a three-bit thread pointer, which could support as many as eight threads. MIPS is probably limiting current implementations of the 34K to five threads while preserving the option of supporting six, seven, or eight threads in future processors.

Each thread requires what MIPS calls a thread context: a separate instantiation of all the structures needed to hold the user-level state information of a process. Each thread context has an independent program counter and a complete set of programmer-visible registers. In the MIPS32 Release 2 instruction-set architecture (ISA), there are 32 general-purpose registers (GPR), 32 bits wide, plus a 64-bit “hi/lo” accumulator for saturating arithmetic. MIPS32-R2 allows developers to implement one or more shadows of each GPR to support vectored interrupts or external interrupt controllers. The DSP extensions carried forward from the 24KE add three more 64-bit accumulators and one 32-bit control register to the ISA. In addition, the 34Kf and 34Kf Pro cores have 32 floating-point registers, 64 bits wide.

With so much user-level state to replicate for each thread context, it’s easy to see why determining the optimal number of simultaneous threads isn’t a

casual decision. Developers working with the 34K need to weigh the performance improvements of multi-threading against the cost of the extra logic (more on this later). An automated processor-configuration tool directs the logic compiler to add the structures necessary for each thread context.



**Figure 1.** In this graphical example of SMT, the top part of the illustration shows three color-coded instruction streams representing different threads or processes. Each thread has gaps caused by cache misses during load operations—gaps that would create unwanted bubbles in a conventional instruction pipeline. The bottom part of the illustration shows how an SMT processor can fetch instructions from different threads to fill the gaps and eliminate the bubbles. SMT requires instant (single-cycle) context switching without flushing the pipeline, which in turn requires duplicate register files to preserve the state of each thread.

**Threading the Needle Pops Bubbles**

Figure 1 sums up the advantages of SMT explained in more depth in previous articles. Normally, a context switch forces a processor to flush its pipeline and begin fetching instructions from the new context. Even without a context switch, pipelines can suffer from bubbles (empty pipe stages) when a slow operation stalls execution. In particular, cache misses create pipeline bubbles when slow external memory forces the processor to wait for a load operation to complete. Another common source of pipeline bubbles is a mispredicted branch instruction. No matter what the cause, bubbles and pipeline flushes waste valuable clock cycles. An SMT processor tries to avoid flushes and fill the bubbles by inserting instructions from different threads.

In addition to the variable number of thread contexts, another design-time option in 34K processors is what MIPS calls a virtual processing element (VPE). This option allows a 34K processor to run two independent operating environments simultaneously, each with its own threads. Each environment could be an embedded OS—either two instances of the same OS or two completely different operating systems. Alternatively, one or both VPEs could support the application software on “bare metal,” without any OS at all. (Many embedded systems don’t need the complexity of an OS or use a custom operating environment.)

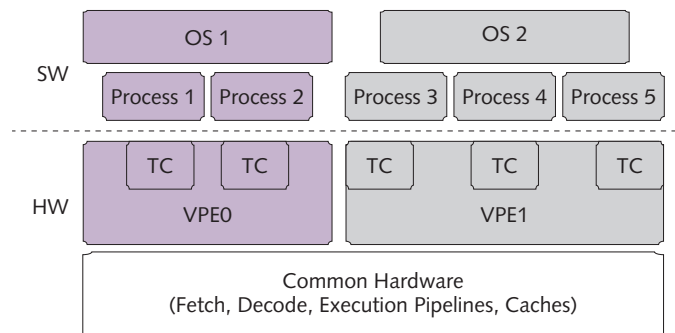
To support each VPE, the processor duplicates all the privileged registers and structures an OS might need. In other words, a VPE holds all the OS-level state information of an operating environment, just as a thread context holds all the user-level state information of a thread. An example of OS-level state is the MMU’s translation lookaside buffer (TLB) or memory map. (As with previous MIPS32 processors, the 34K family allows developers to equip the MMU with either a TLB or a fixed memory map for virtual addresses.)

VPEs provide a simple form of virtualization, much like the virtualization technology now appearing in x86 server and desktop processors from AMD and Intel. (See *MPR 1/30/06-08*, “This Technology Is Virtually Here Now.”) One difference is that embedded OSes need fewer modifications to run within their virtual compartments on 34K processors, because there’s no hypervisor software layer. To

an operating system, each VPE looks like a separate 34K processor. Of course, many embedded applications don’t need VPEs, but some applications can take advantage of this feature. For example, one VPE could run secure processes, such as encryption and electronic commerce, leaving the other VPE to handle unsecure tasks. Or a need for reliability could determine the division of labor.

Figure 2 shows how a typical embedded system might use two VPEs. In this example, one VPE has two thread contexts, allowing it to run two simultaneous threads, while the other VPE has three thread contexts, allowing it to run three simultaneous threads. Although developers must choose the maximum number of VPEs and thread contexts before logic synthesis, the processor can dynamically allocate threads to VPEs at run time. MIPS refers to this dynamic allocation as “binding.”

One cost of using two VPEs is the overhead of their additional logic: about 84,000 gates for both. Another potential cost is larger caches. Both operating environments, each perhaps running multiple threads, must share the same instruction and data caches. Developers choosing the multiple-VPE option should run simulations with realistic workloads to determine if larger caches are necessary.



**Figure 2.** At design time, developers working with a 34K core can choose to duplicate all the resources needed to run an embedded OS on the processor, allowing it to run two operating systems simultaneously. MIPS refers to each set of OS resources as a virtual processing element (VPE). Each VPE can have one or more of its own thread contexts (TC), up to a maximum of five per processor.

Instruction	Description	Comments
EMT	Enable multithreading	—
DMT	Disable multithreading	—
EVPE	Enable VPE	VPE: virtual processing element
DVPE	Disable VPE	—
FORK	Enable thread context	—
YIELD	Disable thread context	—
MTTR	Move to thread register	Write to another thread's registers
MFTR	Move from thread register	Read from another thread's registers

**Table 2.** MIPS32 34K processors add eight new instructions to the MIPS32 Release 2 instruction-set architecture. These instructions allow programmers to enable or disable multithreading, enable or disable the optional virtual processing elements (VPE), enable or disable individual threads, and access registers in different thread contexts.

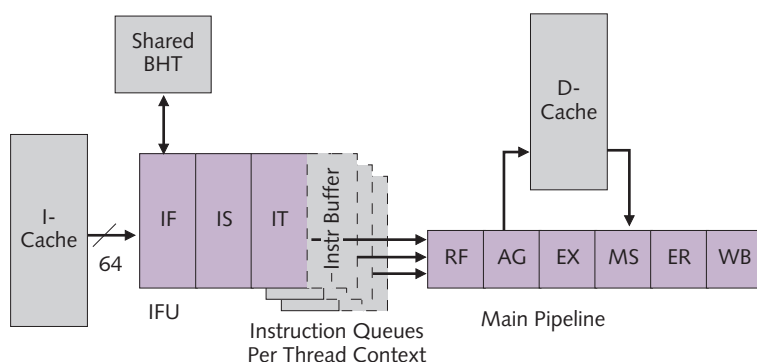
MIPS says normal-size caches are usually sufficient, because the processor can fill the pipeline with instructions from a different thread context after a cache miss.

To prevent cache thrashing—rarely a problem, according to MIPS—programmers can lock individual cache lines or assign individual ways to particular thread contexts. Having the option of higher set-associativity might help in this regard; currently, the 34K limits caches to four-way set-associativity. Another alternative is to stash important code or data in scratchpad RAM, which is a configurable option for all 34K processor cores.

### Thread-Priority Policies Are Configurable

Any SMT processor needs a method of assigning execution priorities to threads. It's particularly important for a multithreaded embedded processor, because embedded programs often have real-time constraints that cannot tolerate thread starvation. To address the needs of as many applications as possible, MIPS allows customers to determine the 34K's thread-priority policy.

The 34K's default thread-priority manager obeys a round-robin policy, which simply allocates an equal number



**Figure 3.** The MIPS32 34K and 24KE processors are close cousins, but the 34K has a slightly deeper pipeline to support simultaneous multithreading. The 34K's extra stage—here labeled IT (instruction-fetch third)—assigns a thread context to the most recently fetched instruction. This assures that instructions from different threads always reference the correct registers and other state information associated with their contexts.

of clock cycles to each thread, one after the other. In a two-threaded implementation of the processor, each thread gets 50% of the available clock cycles; in the maximum five-threaded implementation, each thread gets 20%. (Actually, “thread priority” is an oxymoron with a round-robin manager, because no thread enjoys priority over any other thread.) The round-robin manager is always built into the 34K core.

For more-sophisticated multithreaded applications, the 34K has an optional policy manager that's user programmable. This design-time option adds a mere 5,000 gates to the core, so it's an attractive option. The programmable policy manager allows an OS, the application software, or custom logic to allocate clock cycles to different threads in various predefined ways. Furthermore, software programmers and logic designers can define their own methods of managing and assigning priorities. The programmable policy manager is obviously the most flexible thread-priority solution, because it allows priorities to change dynamically at run time, according to operating conditions. Developers must decide whether the programmable manager is worth the additional gates—the decision is immutable after the design is committed to silicon.

As Table 2 shows, MIPS has added only eight new instructions to the MIPS32-R2 ISA to support multithreading. Most of the instructions are self-explanatory: they enable or disable multithreaded execution, VPEs, or thread contexts. Two instructions (MTTR and MFTR) are special privileged operations that allow a kernel thread to access the registers and other state information of a different thread. (We will describe interprocess communications for user-level software later.) The ISA has no special registers associated with multithreading itself, other than the usual architectural registers for each thread context.

In addition to new instructions, the 34K also has new instruction-dispatch queues and an extra stage near the front of the pipeline. Each thread context has an eight-entry dispatch queue that holds recently fetched instructions. (Branch instructions are decoded at this point, but not most other types of instructions.) These queued instructions wait until the thread-priority manager approves their dispatch to a function unit. The new pipeline stage, inserted after stage 2, assigns each instruction to a thread context. As Figure 3 shows, the 34K pipeline is nine stages deep, compared with eight stages in the 24KE.

What Figure 3 doesn't show is the pipeline detour for MIPS16e instructions. Like almost all 32-bit embedded processors, the 34K has a subset of 16-bit instructions for conserving memory. Supporting 16- and 32-bit instructions in a multithreaded processor caused some headaches for the 34K's designers (“We almost caught our hair on fire,” the engineering director told *MPR*), but they found a solution.

Every fetch from the instruction cache brings 64 bits into the pipeline—either two 32-bit instructions or four 16-bit instructions. If a particular thread is running in MIPS16e mode, it detours through a subpipeline for two extra stages: IR (instruction recode) and IK (instruction kill). In those stages, the processor expands 16-bit instructions into 32-bit operations. The subpipeline rejoins the normal pipeline at the new IT stage. At that point, all the instructions begin entering the thread-dispatch queues.

### Two Methods of Interthread Communication

Sometimes it's necessary for threads to communicate with each other by passing parameters back and forth. However, application-level threads cannot communicate directly, because instructions in one thread context aren't allowed to access registers in another context. (The previously described MTTR and MFTR instructions are privileged kernel-level instructions.) For that reason, the 34K provides two general methods for interthread communication.

One method is already available in all MIPS-compatible processors: the common load-link/store-conditional (LL/SC) instruction sequence. This is a substitute for an atomic read-modify-write operation, which is more often found in CISC instruction sets. In an LL/SC sequence, the processor executes a load-link instruction, which returns the value currently stored at a particular memory location. Next, the processor executes a conditional store instruction, which stores a new value at the location only if another thread hasn't changed the value since the load-link. Using this method, multiple threads can pass parameters by using memory locations as mailboxes.

The drawback of the LL/SC method is that it's not truly atomic—another thread can easily change the value between the moments of the load-link and the conditional store. Programs must repeatedly try to execute the LL/SC sequence within a loop until it completes. When several threads are active, all of them looping in this manner, they can compete with each other and waste numerous clock cycles on their initial load-links.

For that reason, the 34K offers an alternative: an interthread communication unit with lockable mailboxes. This is a configurable design-time option. The mailboxes are memory-mapped 32-bit-wide locations that programs can access with standard load/store instructions. Although mailboxes appear to have conventional memory addresses, they are synthesized as flip-flops in the core. At design time, developers can configure the mailboxes as single-entry slots (like registers) or as FIFO buffers. MIPS provides reference code for up to 16 single-entry mailboxes or up to 16 four-entry FIFOs, but Verilog-savvy developers can create any number of mailboxes or FIFOs they need.

When a thread reads a mailbox (using a normal load instruction), the processor blocks other threads from accessing the mailbox—the memory address is locked. Only one thread at a time can win the lock. When the winning

thread finishes writing a new value into the mailbox (using a normal store instruction), the processor unlocks the mailbox and allows other threads to compete for the lock. This method saves clock cycles over the LL/SC method, because a locked mailbox prevents other threads from executing their initial load-link instructions, so they don't waste time reading a mailbox they're not allowed to modify. In effect, the locking mailboxes bring an atomic read-modify-write capability to the MIPS architecture for the first time.

### Making Performance Trade-Offs

SMT can add almost 200,000 gates to a 34K processor, especially if developers lavishly indulge in all the fancy options: the maximum of five thread contexts, the maximum of two VPEs, a programmable thread-priority manager, an interthread communication unit, and numerous mailboxes. Not to mention all the other options available for the 34K: a sophisticated FPU, CorExtend, a coprocessor interface, scratchpad memory, and TLBs instead of fixed memory maps for the MMUs.

Anyone who has shopped for a new car knows the feeling. The question is whether the higher performance is worth the additional design complexity, silicon, and power consumption. Does the application call for a Lotus or a Prius?

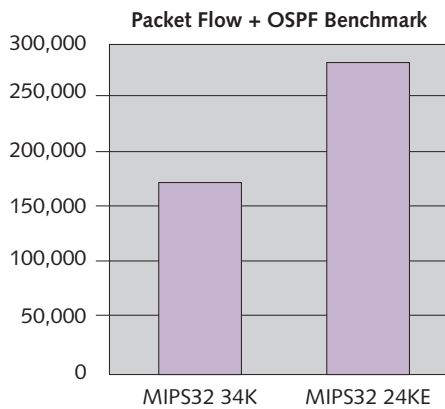
Although the 34K leans toward high throughput, not low power consumption, multithreading adds new twists to the equation. In some embedded applications, a multithreaded 34K processor might replace two or more smaller processor cores, resulting in lower overall system power. This is particularly likely if the 34K has two VPEs, allowing it to run two different operating systems or to strictly isolate two different tasks. Also, keep in mind that the 34K has the same DSP extensions and CorExtend options that the 24KE has. With its signal-processing features and some well-crafted application-specific instructions, the 34K could make a separate DSP or ASIC redundant.

SMT allows a processor to do more work at a given clock speed, so a multithreaded 34K processor could save power over a single-threaded 24KE processor running the same workload at a higher clock speed. Even after accounting for the additional logic SMT requires, the lower-frequency 34K chip could be about the same size as the higher-frequency 24KE chip, because slower memory arrays are denser than faster memories.

Another consideration is that a 34K could save power by using the thread-priority manager to allocate just enough clock cycles for a real-time task, instead of running at a higher frequency to guarantee real-time performance under any conditions. In sum, developers can use SMT to achieve higher overall throughput, lower overall power consumption, or a combination of both goals.

### Measuring the Cost of Performance

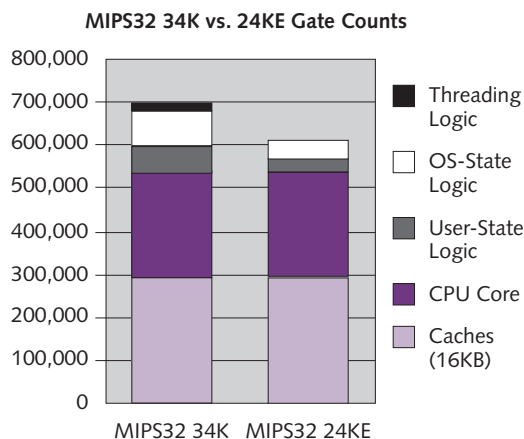
MIPS has some benchmark results favorably comparing the performance of a dual-threaded 34K with a single-threaded



**Figure 4.** MIPS ran a packet-flow test and a packet-forwarding test (open shortest path first) on simulations of similarly configured 34K and 24KE processors. Parallelism is relatively easy to exploit in packet forwarding, which plays to the strength of the multithreaded 34K. This chart shows the number of clock cycles required to complete the tests, so a shorter bar is better—the 34K is 60% faster than the 24KE.

24KE. Naturally, one would expect MIPS to choose an example showing the 34K in the best light, but the benchmarks are interesting nonetheless. For this test, MIPS compared a 24KE processor with a similarly configured 34K processor that has two VPEs and two thread contexts. Both processors had 16KB instruction and data caches. The test, conducted on a cycle-accurate instruction-level simulator, consisted of two packet-processing programs. As Figure 4 shows, the 34K was 60% faster than the 24KE.

Another interesting result of this test was that the 34K missed the cache slightly more often than the 24KE did. The 34K's miss rate was 5.16%, compared with 4.41% for the 24KE. (Those miss rates are much lower than would be expected when running real-world software, probably because there wasn't enough packet data to fully exercise



**Figure 5.** This chart shows the number of gates required for two similarly configured 34K and 24KE processor cores—the same configurations MIPS used to obtain the benchmark results in Figure 4. With two thread contexts and two VPEs, the 34K core has 14% more gates than the single-threaded 24KE, but it's 60% faster in the packet-processing tests.

the memory system.) Despite the 34K's slightly higher miss rate, it managed to execute 0.61 instructions per cycle (IPC) in this test, compared with only 0.37 IPC for the 24KE. These statistics suggest two conclusions. The 34K probably missed the cache more often because its two threads shared a cache no larger than the single-threaded 24KE's cache (16KB). But multithreading worked as intended: when the 34K suffered a cache miss, it was able to execute instructions from its other thread.

CPU architects can do many things to improve the performance of a processor, but is the improvement worth the extra gates? In this (admittedly limited) example, the answer is yes. The dual-threaded 34K processor was 60% faster than the single-threaded 24KE processor, but it requires only about 14% more silicon. This estimate is based on gate counts extracted from post-layout models of both cores as configured for the packet benchmarks. Using die-area data that MIPS provided for a 0.13-micron fabrication process, *MPR* calculates that this configuration of the 34K core has 694,000 gates, and the similarly configured 24KE core has 607,000 gates.

Note that caches alone account for the area-equivalent of 290,000 of those gates, and the common hardware in the CPU cores is 250,000 gates. The important data is the extra logic required to support multithreading. *MPR* estimates that the threading logic (such as the thread-policy manager and interthread communication unit) adds 14,000 gates; the structures required for OS-level state information in two VPEs add 40,000 gates; and the structures required for user-level state information in two thread contexts add 23,000 gates. Figure 5 shows a breakdown of the gates in the 34K and 24KE cores, based on their configurations for the packet benchmarks.

### Alternative Paths to High Performance

With the 34K processor, MIPS is taking a very different path to higher performance than other processor-IP vendors are taking. The 34K is the world's first licensable multithreaded embedded-processor core, and it's also the most configurable core available from MIPS. In contrast, ARM's highest-performance processor core, the new Cortex-A8, bets on two-way superscalar pipelines instead of multithreading. ARC and Tensilica are strong believers in user-defined extensions—a capability available in some MIPS processors, including the 34K Pro series, but not as heavily promoted by MIPS. All four companies have customers using multi-core designs to reach higher performance.

At first glance, it's surprising that MIPS has chosen multithreading over superscalar execution. Today's embedded MIPS processors are descendants of the MIPS workstation/server processors of the late 1980s and 1990s. MIPS introduced its first single-chip superscalar processor, the R10000, in 1995. The R10000 was a four-way out-of-order design with speculative execution, still impressive 11 years later. Considering that history, one might expect MIPS to

use the familiar technique of superscalar pipelining instead of a less popular technique like multithreading.

However, MIPS concludes that SMT is a more gate-efficient path to higher performance than superscalar execution is. Perhaps this reflects MIPS's greater experience with superscalar design. (The Cortex-A8 is ARM's first superscalar core.) Both approaches have significant overhead in extra logic—duplicate pipelines for superscalar, duplicate register files for multithreading. But multithreading can deliver more bang for the buck than superscalar can, especially in an embedded processor. And the ability to switch contexts in a single clock cycle without flushing the pipeline has obvious advantages in real-time systems.

Multicore processors have even more duplication than superscalar or multithreaded processors, because they replicate entire cores, not just pipelines or registers. MIPS says a multithreaded processor can beat a multicore design—to a point, at least—after accounting for the extra die area and power consumption of a multicore chip. Both approaches rely on finding enough parallelism in the software to justify the additional hardware.

Table 3 compares the MIPS32 34K processor with the MIPS32 24KE, ARC 700, ARM Cortex-A8, Tensilica Xtensa 6, and Tensilica Xtensa LX. All are licensable 32-bit embedded-processor cores, and all are the highest-performance examples of their architectures. If low power is more important than high throughput, ARC, ARM, and MIPS offer much smaller cores than those shown here, and a minimal configuration of Tensilica's cores will serve the same purpose. We derived the core sizes and power numbers in the

## Price & Availability

MIPS Technologies is licensing the MIPS32 34K family of processor cores now. The four members of the family are the 34Kc, 34Kf, 34Kc Pro, and 34Kf Pro. All are available as synthesizable IP in Verilog. Like most processor-IP vendors, MIPS doesn't disclose up-front license fees or chip royalties, which are negotiable. For more information about the MIPS32 34K family, visit [www.mips.com/content/Products/Cores/32-BitCores/MIPS3234K/ProductCatalog/P\\_MIPS3234KFamily/productBrief](http://www.mips.com/content/Products/Cores/32-BitCores/MIPS3234K/ProductCatalog/P_MIPS3234KFamily/productBrief).

table from the best available vendor data, but take them with a grain of silicon—there are too many variables to support quick conclusions. With the exception of the Cortex-A8, all these cores have numerous configuration options that greatly affect their size and power.


Multithreading makes the 34K unique among licensable processor cores. But customers want performance, not novelty. Last year, ARM encroached on MIPS's high-performance territory with the superscalar Cortex-A8, and now MIPS is responding with the multithreaded 34K. Without a doubt, superscalar pipelining is the better-understood technology. Even undergraduate computer-science students write compilers for superscalar microarchitectures. Simultaneous multithreading is less understood and poses

Feature	MIPS MIPS32-34K	MIPS MIPS32-24KE	ARC ARC 700	ARM Cortex-A8	Tensilica Xtensa 6	Tensilica Xtensa LX
Architecture	MIPS32-R2	MIPS32-R2	ARCCompact	ARMv7	Xtensa	Xtensa
Pipeline Type	In-order uniscalar, 5-way threading	In-order uniscalar	In-order uniscalar	In-order 2-way superscalar	In-order uniscalar	In-order uniscalar
Pipeline Depth	9 stages	8 stages	7 stages	13 stages	5 stages	5 or 7 stages
Branch Predict	Dynamic	Dynamic	Dynamic	Dynamic	—	—
L1 I-Cache	0–64K	0–64K	0–64K	16–32K	0–32K	0–32K
L1 D-Cache	0–64K	0–64K	0–64K	16–32K	0–32K	0–32K
L2 Cache	—	—	—	256K	—	—
L3 Cache	—	—	—	Optional	—	—
Configurability	Optional; high	Optional; medium	High	Low	High	High
Instr Length	32 bits	32 bits	32 bits	32 bits	24 bits	24 bits
Short Instr	16 bits	16 bits	16 bits	16 bits	16 bits	16 bits
DSP Extensions	Yes	Yes	Optional*	Yes	—	Optional
Java Extensions	—	—	—	Yes	—	—
FPU	Optional 32/64 bits	Optional 32/64 bits	Optional 32 or 64 bits	Optional 32/64 bits	Optional 32 bits	Optional 32 bits
MMU	Yes	Yes	Yes	Yes	Optional	—
Core Freq	500MHz‡	400†	400MHz	600MHz–1GHz**	350–400MHz†	350–400MHz†
Core Size (Base)	2.1mm <sup>2</sup> (90nm)	3.0–4.8mm <sup>2</sup> (130nm)	95k gates	<3mm <sup>2</sup> (65nm)	20k gates	20k gates
Power/MHz	0.56mW‡	0.58–0.61mW†	0.15mW†	0.5mW**	0.04mW†	0.04mW†
Introduction	2006	2005	2004	2005	2005	2004

**Table 3.** All these processors are high-performance IP cores based on 32-bit RISC architectures. Their pipelines reveal the starkest differences. Most processors in this class have simple uniscalar pipelines, but MIPS has introduced the first core with simultaneous multithreading, whereas ARM is aiming for high performance with two-way superscalar execution. ARC and Tensilica rely more heavily on user-defined application-specific extensions. Core-size and power-consumption numbers are vendor estimates, based mostly on simulations. \*The ARC 700 has some built-in DSP instructions; more powerful extensions are optional. †Assumes a generic 130nm fabrication process. ‡Assumes a generic 90nm process. \*\*Assumes a generic 65nm process.

a greater challenge for programmers wanting to make the most of the 34K's best capabilities.

However, multithreading is more versatile than superscalar execution. As implemented in the 34K, it allows a processor to execute multiple threads with strict task isolation,

to run multiple operating systems, to instantly switch contexts, and to precisely allocate clock cycles among various tasks—extremely valuable capabilities for embedded applications. The 34K is unique for a purpose. It's a significant advance for embedded-processor cores. 

*To subscribe to Microprocessor Report, phone 480.483.4441 or visit [www.MPRonline.com](http://www.MPRonline.com)*

