Reed Electronics Group

MICROPROCESSOR REPORT

www.MPRonline.com

THE INSIDER'S GUIDE TO MICROPROCESSOR HARDWARE

EXPLORING THE ARM1026EJ-S PIPELINE

Extensive Architectural Modeling Highlights Frequency and IPC Tradeoffs

By Markus Levy {4/30/02-01}

Whenever a processor core is converted from a hard macrocell to a synthesizable component it loses performance. Wrong! At **Embedded Processor Forum 2002**, ARM announced details of the new ARM1026EJ-S, the soft-core descendant of the semicustom ARM1020E core. In addition to making this new core fully synthesizable and incorporating ARM's Jazelle technology (Java accelerator), one of ARM's primary goals was to ensure that its instructions per clock (IPC) numbers were as good as, or better than, those of the 1020. Another goal was to allow customers to use compiled RAM macros and still comply with the critical timing paths between the core and caches. There are also some critical paths to the MMU, but these have a minimal effect, because the RAM used for the TLBs is smaller and faster, and the TLB entries that were most recently used are cached in the µTLBs inside the core. To meet these goals, the 1020's pipeline went in for a tune-up, to include performance enhancements to rearrange certain elements of the datapath, in order to reduce critical synthesized timing delays resulting from higher clock frequencies.

Eric Schorn, ARM's CPU product manager, presented the architectural and benchmarking details for the ARM1026EJ-S at EPF 2002.

ARM10 Slowly Gaining Momentum

The ARM10 microarchitecture was first announced at the 1998 Embedded Processor Forum, albeit very early in the product-development stages (see *MPR 11/16/98*, "ARM10 Points to Set-Tops, Handhelds"). Although the ARM10 family now has seven licensees, which include Agere, Samsung, TI, TSMC, and UMC, one might still say that the architecture's time hasn't yet come. ARM claims it is "in licensing discussions with several major semiconductor companies for the ARM1026EJ-S and the core will be generally available within a few months." However, the bulk of ARM's business is now shifting from ARM7 to ARM9 core families, and customers should soon begin taking advantage of the greater performance and bandwidth benefits of the ARM10 (see Table 1).

Initially, ARM's design of the 1020E targeted the 0.18µm and 0.13µm processes. Although this hard macro is designed to work with a variety of design rules, it is obviously less flexible than a soft design that works with synthesis and automatic-place-and-route tools. From a business perspective, a synthesizable version of the 1020 core also makes sense, as ARM's customers are flocking to the synthesizable ARM926EJ-S and ARM946E-S. Unlike ARM's other synthesizable cores, however, the 1026 has enjoyed the luxury of a major redesign that will allow it to maintain the same performance levels as the 1020—or better ones. For example, the ARM940T and 946E-S differ in operating frequency by about 25%. Additionally, the 1026 includes support for Java hardware acceleration, making this core more enticing than the 1020 for many wireless applications.

	ARM 920T	ARM 926EJ-S	ARM 1022E	ARM 1026EJ-S
Instruction Set Version	V4T	V5TEJ	V5TE	V5TEJ
Features	MMU, 32-bit AHB	TCM, MMU, dual AHB, Jazelle, DSP	MMU, dual AHB, DSP	MMU/ MPU, dual AHB, DSP, Jazelle, vector interrupt
Cache Size (I/D)	16K/16K	Variable	16K/16K	Variable
Pipeline Stages	5	5	6	6
Clock Speed (worst case @ 0.13μ)	250	220–250	266–325	266–325
Clock Speed (typical @ 0.13μ)	375	330–375	400–475	400–475
Typical Power (mW/MHz w/ 16K/16K caches)	0.36	0.4	0.6	0.5
Area (16K/16K caches, mm²)	4.7	3.75	6.9	4.6

Table 1. Comparison of the 1026 with other hard and soft ARM processor cores. Note that although the 1020 is referenced throughout this article, the 1020 and 1022E differ only in cache sizes (32K and 16K, respectively). The 1026 is incrementally more expensive than the 926, demonstrating ARM's desire to move its ARM9 customers to the newer architecture.

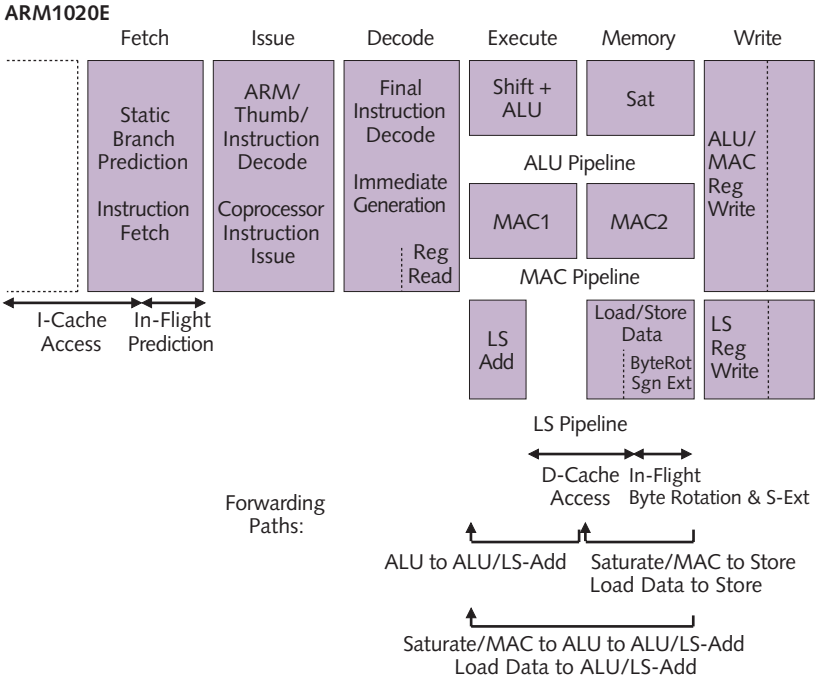

Figure 1. The pipeline and basic microarchitectural structure of the 1020 depicts the six-stage pipeline and some of the critical timing components.

There is also a benefit to using compiled RAM for the caches. On some processes, the compiled RAM caches are about half the area of portable custom caches, because these can be generated for a specific target process rather than being ported from generic rules. However, compiled RAM also has drawbacks. Depending on the process, the routing between all the RAM blocks of a compiled cache and the core can have a significant affect on area. Specifically, however, the compiled RAM tend to be designed for high density rather than for high speed. This situation can lead to serious headaches for engineers trying to couple the RAM to a high-performance core without having an impact on the maximum system frequency.

Improved Performance Insight

To begin the tune-up process for the 1026, ARM modified the 1020 architectural simulation model, ensuring that its base configuration was represented by a new C-like model that was almost identical to the previous RTL model (on a cycle-by-cycle basis). The next step in the process was to obtain representative code to use as benchmarks. Historically, the lack of application-specific benchmarks has made design simulation comparison difficult for the entire embedded industry. However, the EEMBC benchmarks have become the industry-standard suite that covers many of the key market segments, including telecom, networking, consumer, automotive/industrial, and office automation. ARM engineers made effective use of these benchmarks in their extensive analysis of design tradeoffs that could be implemented in the new 1026. In fact, one side benefit of this overall effort was the EEMBC Certification Lab (ECL) certification of the 1020 design across all five EEMBC suites (full results available at *www.eembc.org*).

Three sets of interacting design factors influenced the conversion of the 1020 to the synthesizable 1026. The first set of factors that had to be addressed related to the way to make a synthesis-friendly processor core. These include converting the 1020's latch-based datapath to an edge-triggered flip-flop design; removing all dynamic logic and tristate buses; and partitioning the design to optimize the use of EDA synthesis tools.

The second set of variables includes those design changes that decrease IPC but benefit frequency. One example is the creation of a two-cycle memory access to accommodate the compiled RAM; the 1020 requires only a single cycle. The third set of variables consists of those variables that increase IPC (without decreasing frequency) and are functional enhancements to the core. Such modifications include an improved, yet still simple,

branch prediction mechanism and a return-from-subroutine stack. ARM also reduced the core to AMBA high-speed-bus (AHB) latency to improve bandwidth when it is operating with applications having large datasets.

Compiled RAM Requires More Time

The most obvious change that would have to take place to convert the ARM-1020E to a synthesizable design is to extend the timing for the instruction and data caches (indicated by the arrows for I-cache and D-cache access in Figure 1). The semicustom memories of the 1020 allow the processor to use 1.5 cycles from address-out to data-return, plus the byte-rotate portion. (The slower ARM9T allows one cycle.) ARM1020E and the most recent generations of ARM9 use a dedicated adder (Load/Store Add, or LS-Add) to generate the address for data at the end of phase 1 of the execute stage (output from the LS-Add block). This timing allows for forwarding from an ALU instruction that produced its result in the previous cycle (i.e., there's no penalty for using the result of an ALU operation in an address calculation for the very next instruction).

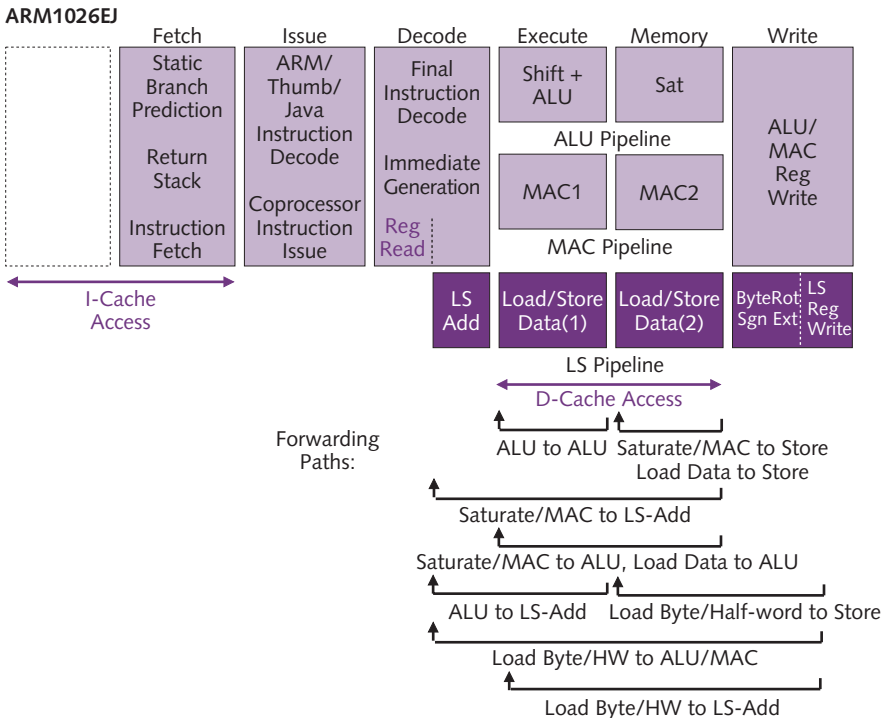

Figure 2. The highlighted areas (dark purple) in this 1026 diagram show the major pipeline changes to accommodate compiled RAMs.

Accessing the L1 caches is high on the list of critical paths for most microprocessors, so it's not surprising that trying to hook up the caches is often a major timing issue for engineers working with synthesizable cores. In the 1026, ARM ameliorates the situation by allowing two cycles for memory accesses (Figure 2). The RAM must still operate in one cycle, but it gets to use far more of the cycle, because there is time to propagate the address out to the RAM before the access, plus subsequent time to propagate data back to the core after the access. These propagation delays become more important in larger caches (greater than 16K) and new process generations, where interconnect delays are becoming increasingly significant

The extra half-cycle tagged onto the cache access is implemented by pushing address generation earlier rather than by allowing data to return later. Consequently, there is no extra penalty for data that is loaded and then used by an ALU operation, but fewer sources can forward data to address generation. An extra cycle is needed when loaded data is immediately used in an address calculation for a load/store by a subsequent instruction, because the data is not available to forward until the write stage.

Architectural analysis of the extra data-cache-access cycle required real-world assessment to determine its impact on overall core performance. Selecting from a variety of the EEMBC benchmarks, ARM designers were able to measure the effect of the extra data-cache-access cycle. The performance penalties when running different application benchmarks are shown in Figure 3.

The ARM instruction set includes loads and stores that perform a shift as part of the address calculation. Data address calculations for the 1026 that require that the full shifter (e.g., *LDR R0, [R1, R2 LSL #3]*) take an extra cycle in

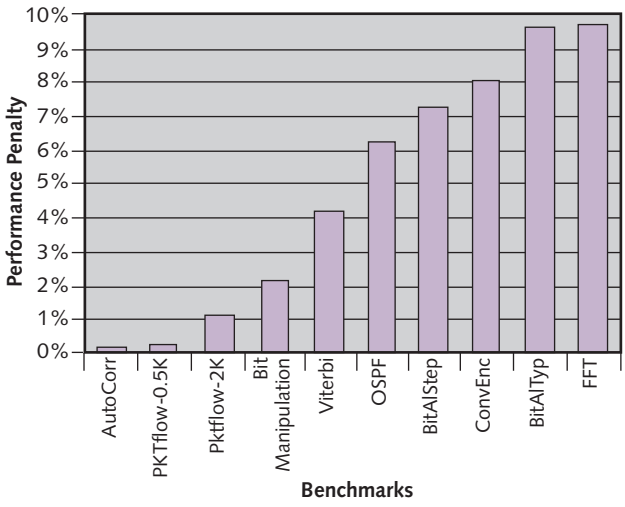

Figure 3. Example performance penalties associated with the extra cycle for data-cache access. Although many of the benchmarks experience minimal or no effect, others experience a substantial hit.

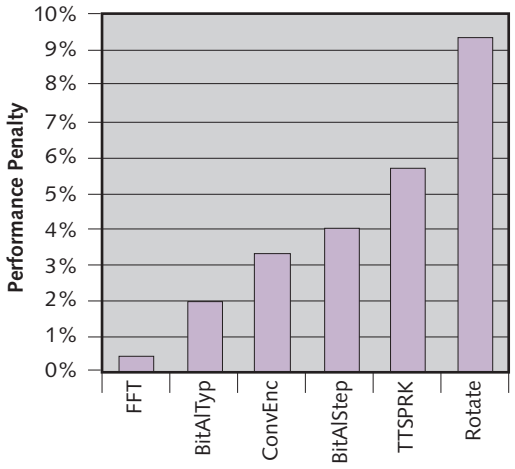

Figure 4. The performance penalty of one additional cycle (beyond the 1020) associated with mispredicted branches.

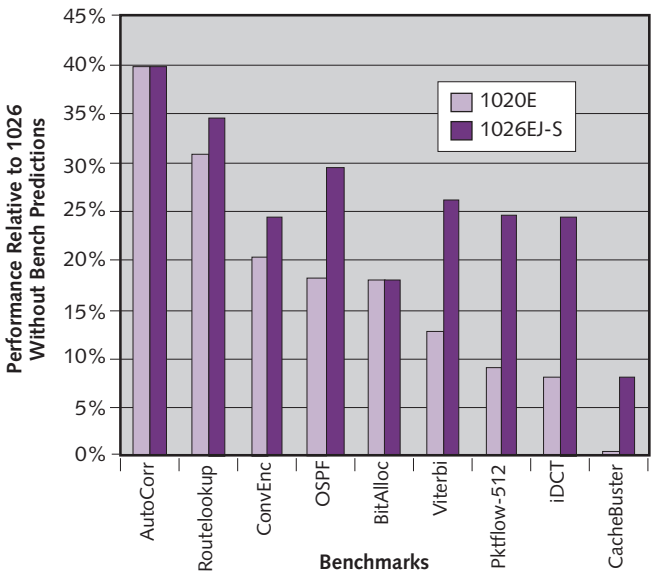

Figure 5. Comparing the branch prediction mechanisms of the 1020 and 1026 with those of the 1020 minus branch predictions shows that some benchmarks improve significantly by using the basic branch prediction capability of the 1020. Other benchmarks, such as those with many sequential (or nearby) branches in the code, demonstrate the benefits of the 1026's enhanced branch prediction scheme. Not depicted in this graph is the fact that the 1020's branch prediction mechanism yielded (a) the most benefits on average for the majority of the benchmarks and (b) the most extreme improvements, with quite a few benchmarks showing 15–40% increases.

the 1026, compared with the 1020. However, modeling indicated that adding a simple multiplexer in the decode stage would allow the core to handle shifts of one or two places and minimize the negative effects of the extra access cycle.

No More Missed Predicted Branches

On the instruction cache side, the effects of the extra RAM-access cycle result in a four-cycle penalty for branch mispredicts and indirect branches (an increase of one cycle) for the ARM1026EJ-S, because a new instruction address is not available until the mispredicted branch arrives at the start of the memory access cycle. Therefore, the first five stages of the pipeline must be refilled, resulting in the four-cycle penalty.

With the 1026's improved branch prediction, the EEMBC benchmark analysis shows negligible effect from the extra cycle taken by indirect branches that cannot be predicted with a static predictor. The extra cycle incurred by branch mispredicts is more significant and resulted in an average penalty of 2–3%, but, in one case, the penalty was as high as 9.3% (Figure 4).

Whereas ARM9 family processors assume that all branches are not taken, the 1020 and 1026 implement a simple static predictor (backward taken, forward not taken) that achieves a hit rate of about 65–70% in many cases. The most significant performance improvement of the 1026 is due to an enhancement of the 1020's branch prediction detection circuitry attached to the processor's prefetch buffers. The 1020's branch predictor made two significant compromises to minimize its areas: it allowed only one unresolved prediction at a time, and it implemented only one predictor and target adder. It would therefore occasionally be unable to predict a branch, either because a prediction was already in the pipeline, or because another instruction was occupying the prefetch slot associated with the predictor.

For some extremely tight code loops, or loops containing groups of branches, the 1020 predictor can miss a critical branch on every iteration (Figure 5). The 1026's branch predictor eliminates these unpredicted branches. The benchmarks where the 1020 predictor was not predicting a critical branch are the ones that show the most improvement with the 1026 predictor. Benchmark results indicate that this is the most important architectural enhancement and is the most significant factor in balancing the negative effects on CPI introduced by the datapath changes required to make the core synthesizable. This modification added about 20% gate-count overhead to the branch prediction logic of the 1026.

Sequential Access Detection

Streaming sequential data is one of the common operations of multimedia applications. Video, for example, performs strings of byte-store operations that bypass the data cache and go directly to/from the frame buffer. JPEG is similar, in that it reads sequential bytes (pixels) as it compresses or decompresses the data. Prior-generation ARM processors marked buffered stores as sequential transfers on AHB only when those stores were part of a cache line eviction or a store multiple instruction. Essentially, this action transforms store operations from sequential to nonsequential and results in six bus cycles per store (6-6-6-6) instead of one bus cycle in a typical "6-1-1-1" memory system.

To overcome this type of sequential data-store overhead in the ARM1026, ARM added address comparators to

the AHB write buffer to detect sequential, noncachable stores and form them into AHB bursts. Most of the EEMBC benchmarks do not operate with this type of streaming data, but benchmarks such as FFT, JPEG, and RGB-to-CMYK conversion benefit tremendously from this architectural modification. (Performance improvements are approximately 5%, 19%, and 64%, respectively.)

The ARM compiler tries to use store-multiple instructions, but this is not possible in many cases: for example, when stores to adjacent addresses use different registers in their address calculation or for multiple byte stores (there is no store-multiple-byte instruction). It is also difficult for the compiler to merge stores from loops that perform one store per iteration.

Core-to-AHB Latency Decreases

Several ARM licensees were designing the ARM1020E into systems that implemented very low core-to-AHB clock ratios. The effects of core clock cycles in the line-fill latency are more exaggerated in systems having lower bus ratios, compared with systems having a core-clock frequency that is significantly faster than their bus-clock frequency, because significantly fewer overall core cycle counts are involved.

ARM engineers determined that the line-fill latency could be reduced by five clock cycles. Two of these clocks came from placing the AHB in the same clock domain as the core. The synthesizable 1026 is clocked using an integer multiple of its AHB clock. The AHB bus segment that is running off the processor core's clock can be bridged to another asynchronous AHB bus segment, if required. The remainder of the cycle savings came from optimizations to the logic between the core and AHB.

The benefit of using a synchronous interface is most obvious on applications that have large datasets. One example is the booting of an operating system. In addition, a few of the EEMBC benchmarks have datasets that are large enough to stress the memory system.

ARM Pushes Sign Extension

On algorithms or benchmarks that operate on bytes or halfwords, data being loaded can be rotated and aligned and sign- or zero-extended before being used by a subsequent instruction. In the ARM1020E, this operation takes place partly at the end of the memory cycle and partly in conjunction with the operation of the pipeline's forwarding multiplexers (Figure 1). Within the synthesized logic of the 1020, the sign extension must be pushed out into the write stage of the pipeline, forcing an extra cycle to be taken when the data is needed by an instruction following closely behind the load (Figure 2). Furthermore, this action also reduces the number of forwarding paths for data returning from the data cache.

The negative effect of this timing modification is most dramatically witnessed by running the EEMBC Telecomm benchmarks with a performance reduction averaging 9.5% across the suite, minimizing many of the gains from the branch predictor enhancements (at least for the Telecomm benchmark suite). To be true to the DSP algorithms, all the input and output data are maintained in their 16-bit original format. Hence, the processor will automatically sign-extend all 16-bit load data.

Single-Entry Return Stack

Despite the obvious benefits of the ARM1026's improved branch prediction mechanism, a mispredicted branch is still costly and occurs, on average, 30% of the time. However, a subroutine return is one form of a branch instruction that can be accurately predicted most of the time and thereby avert the infamous pipeline flush. In the ARM architecture, a program calls a subroutine by issuing either a *BL <Imm>* or *BLX <Imm>* (branch-and-link or branch-and-exchange) instruction. At the end of the subroutine, a *MOV PC, R14* (or *BX R14*) instruction restores the program counter. These instructions, which are a form of indirect branches, cause the value stored in special function register R14 to be loaded into the program counter (PC).

Modeling determined that a return stack would yield a performance boost because it helps avert pipeline flushes on returns from subroutines. By "remembering" the PC of an instruction after a branch for *BL <Imm>* or *BLX <Imm>,* the return stack predicts using the stored value for R14 on *MOV PC, R14* or "*BX R14*." In the 1020, indirect branches were not predicted, so they took four cycles; with the 1026, the indirect branches take five cycles, but at least now, indirect branches such as *MOV PC, R14* are predicted. Because subroutine returns dominate indirect branches, the benchmarks demonstrate an average improvement of 1.1%. Most EEMBC benchmarks are not laden with subroutines; however, the networking and automotive suites contain benchmarks that improve between 2% and 10% from this return stack.

Modeling also determined that much of the benefit of having a return stack is gained by implementing a single-entry stack. From a hardware perspective, a single-entry stack is much easier to implement than a multiple-entry stack. The processor need only monitor the instructions *BL R14 and BLX14* to determine when to push onto the stack and *MOV PC, R14* and *BX R14* to determine when to pop the stack. For a multiple-entry stack, loads from the program's memory stack must be considered as well, which means branch mispredict checking must occur in both the execute and memory stages of the pipeline.

Seeing the Forest Through the Trees

Few architectural decisions are made by evaluating the performance of individual benchmarks. Nevertheless, the preceding analysis pointed out the effects of IPC and frequency design tradeoffs on individual benchmarks to help designers spot trends and/or anomalies that could be resolved. However, final implementation of the 1026 was based on an understanding of the big picture using the averages from

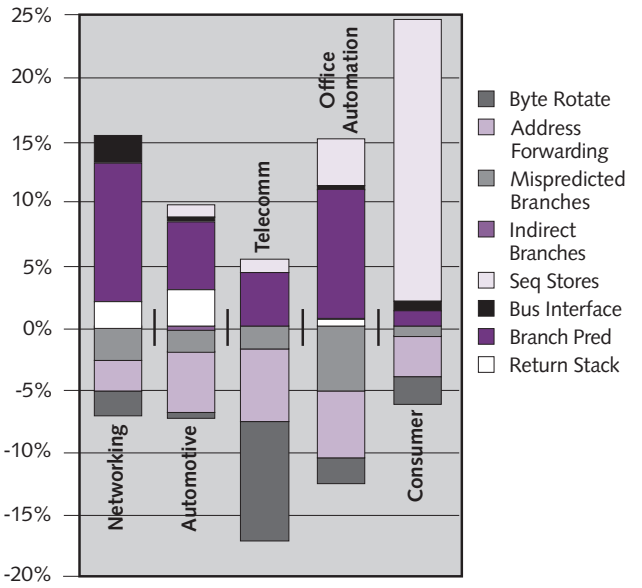

Figure 6. Benchmark results depict the overall tradeoffs of converting from a hard macrocell to a synthesizable component.

each of the EEMBC application areas, as well as other functions such as the booting of an operating system (Figure 6).

Similar to many intellectual property core vendors, ARM is reluctant to hand out pricing, as the core licensing model is such an inexact science. Therefore, assuming that the price is right, *MPR* believes that the 1026 offers enough compelling reason to convert customers from the ARM9 family. The 1026 should also give ARM customers better ammunition to compete against Intel's XScale, especially for those implementing systems with Java support. However, for a more reasonable head-to-head against XScale, ARM's customers should probably hold out for ARM11, although this would certainly delay the battle. Further, the hand-crafted XScale will be tough to beat on performance and power.

The bottom line is that it was good to see the detailed level of engineering and analysis that ARM provided for its 1026 product development. Although the design modifications of the 1020 to get to the 1026 were not trivial, the architectural modeling helped to simplify the decisions for the design tradeoffs.

110 Fulbourn Road
Cambridge
CB1 9NJ
England
Tel: (44) 01223 400400
Fax: (44) 01223 400410
www.arm.com

Subscription Information

To subscribe to *Microprocessor Report*, contact our customer service department in Scottsdale, Arizona by phone, 480.609-4551; fax, 408.737.2242; email, *emckeighan@instat.com;* or Web, *www. MDRonline.com.*

	U.S. & Canada*	Elsewhere
One year		
Hardcopy or Electronic	$695	$795
Both Hardcopy and Electronic	$795	$895

	U.S. & Canada*	Elsewhere
Two years		
Hardcopy or Electronic	$1,295	$1,495
Both Hardcopy and Electronic	$1,395	$1,595

**Sales tax applies in the following states: AL, AZ, CO, DC, GA, HI, ID, IN, IA, KS, KY, LA, MD, MO, NV, NM, RI, SC, SD, TN, UT, VT, WA, and WV. GST or HST tax applies in Canada.*