Cahners
MICROPROCESSOR REPORT
www.MPRonline.com
THE INSIDER'S GUIDE TO MICROPROCESSOR HARDWARE

IMSYS HEDGES BETS ON JAVA

Rewritable-Microcode Chip Has Instruction Sets for Java, Forth, C/C++

By Tom R. Halfhill {8/14/00-04}

Depending on your point of view—and there seems to be no middle ground here—microprocessors that natively execute Java bytecodes are as palatable as latte or as loathsome as stained teeth. But in Sweden, where the spirit of neutrality still flourishes, a company called Imsys hasn't stopped trying to accommodate both sides by offering an embedded processor with rewritable microcode that natively runs Java or doesn't, as you please.

When last we reported on Imsys, the fabless semiconductor company was on the verge of shipping its GP1000, which has a ready-to-run microcode library that natively executes Java bytecode instructions (see *MPR 12/28/98-03*, "GP-1000 Has Rewritable Microcode"). The GP1000 made it out the door and found some customers, although its most prominent design win—a color-printer engine made by another Swedish company, Array AB—doesn't use the chip's Java capabilities. Imsys says some other customers are interested in using the GP1000 as a Java chip, but they aren't ready to publicly announce their products.

Encouraged by the GP1000's success, Imsys is introducing an enhanced version of the processor known as the Cjip—which is pronounced "chip," even though it substitutes a Java-style "j" for the "h." (Presumably, this works better in Swedish.)

The most interesting new feature of the Cjip is that Imsys has developed an entirely new instruction set to supplement the Java instruction set, which has also been improved. The new instruction set, available as a microcode library, supports Forth or C and C++, using a Java-like stack architecture. This leverages the Cjip's stack memory and, says Imsys, allows the C and C++ compilers to generate much denser code. To prove the point, Imsys has developed an embedded Linux operating system for the Cjip that requires only 650K of memory.

The Cjip is similar to the recently announced aJ-100 Java processor from aJile Systems (see *MPR 8/7/00-02*,

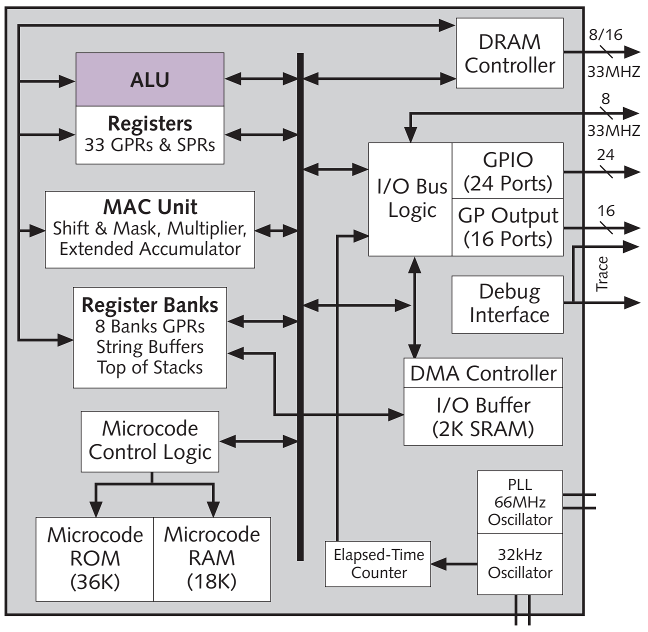

Figure 1. The Cjip is a microcoded CISC processor designed for stack- or register-based programming languages. Two-thirds of the microcode is immutable, but the other third is stored in 18K of SRAM that's rewritable even after bootup.

"Embedded Java Chips Get Real"). Both companies have implemented critical parts of Java's virtual platform in microcode on stack-based embedded processors. Another competitor is Patriot Scientific's PSC1000, which also has a stack-based architecture that supports Java, C, or Forth (see *MPR 4/15/96-01*, "New Embedded CPU Goes ShBoom").

All three companies seek customers that want to write embedded applications in Java but have been deterred by the relatively slow performance, large memory requirements, and nondeterministic memory management of Java bytecode interpreters, just-in-time (JIT) compilers, and Java virtual machines (JVMs). Imsys and Patriot are less religious about Java than aJile and offer alternative instruction sets for customers that feel likewise.

Lower Power Consumption

Fundamentally, the 16-bit Cjip is little changed from the GP1000. As Figure 1 shows, the Cjip retains all the GP1000's unusual features, including a rewritable microcode store, microcode-level concurrency, eight register banks for fast context switching, on-chip stack and string buffers, a DMA controller, a DRAM controller, and a multiply-accumulate (MAC) unit. The die is only 16mm^2 and is packaged in a 144-pin TQFP.

As with the latest version of the GP1000, the Cjip nominally runs at 66MHz. The Cjip can maintain that clock frequency over a voltage range of 2.7–3.6V. Boosting the lower-voltage limit to 3.0V allows for increasing the clock frequency to 80MHz and the DMA capacity from 33MB/s to 40MB/s, although that would require faster DRAMs (50ns instead of 70ns). While the Cjip retains the integrated DRAM controller found on the GP1000, it still works only with EDO DRAM; using SDRAM (for availability reasons) would require some external logic and wouldn't be any faster. The DRAM can be 8 or 16 bits wide, and the Cjip can address up to 128MB.

One of the Cjip's most significant improvements over the GP1000 is that it consumes 50% less power—without the benefit of a process shrink. (The Cjip is still fabricated in a 0.35-micron CMOS process by Ericsson Microelectronics.) To achieve this, Imsys fixed a simple design flaw discovered late in the development of the GP1000. In that processor, the 36K of microcode ROM and 18K of microcode SRAM both draw power at the same time, even though only one memory is accessed each cycle. By alternating power between the two memories and making a few other improvements, Imsys slashed the Cjip's power consumption to 165mW, compared with 330mW for the GP1000. And that's 165mW when running a worst-case microcode loop at a nominal voltage of 3.3V, so typical power consumption (though not yet measured by Imsys) should be even less. At a nominal voltage of 3V, worst-case power consumption drops to 135mW. In fact, the 66MHz Cjip consumes less power than the original 33MHz GP1000.

For Imsys, rewritable microcode isn't just a way to support multiple instruction sets. The Cjip also uses microcode to manage memory and low-level concurrency, to add application-specific instructions, and to implement "veripherals" (virtual peripherals). A control program written entirely in microinstructions handles multitasking at the microcode level, almost like a small RTOS. It manages interrupts, I/O, and priority scheduling for multiple microcode processes. Special microcoded instructions created for Imsys's printer customers can perform such high-level tasks as halftone screening, adaptive thresholding, and RGB-to-CMYK color-space conversions.

Microcoded veripherals are substitutes for small peripherals that normally would be integrated on chip or added as external devices. For instance, one veripheral is a timer that can activate interrupt routines with 60-microsecond resolution. Other veripherals in the Imsys library are watchdog timers, a real-time clock, an LCD controller, keyboard/keypad controllers, sound generators, UARTs, an I^{2}C interface, an IEEE-1284 parallel interface, a multimedia card interface, a stepping-motor controller, a finite-impulse-response (FIR) filter, fax compression/decompression algorithms, and an Ethernet device driver. Imsys will create additional veripherals to customer specifications. Veripherals that require an off-chip interface use some of the Cjip's 24 general-purpose I/O (GPIO) ports or 16 general-purpose output ports.

Triscend takes a somewhat similar approach to soft-peripheral integration (see *MPR 11/16/98-02*, "Triscend E5 Reconfigures Microcontrollers"). The main difference is that Imsys writes the veripherals in microcode for a proprietary 16-bit architecture, while Triscend implements peripherals in programmable logic around an industry-standard 8-bit architecture (an 8032-compatible core). Also, Imsys does not expose microcode programming to customers, while Triscend provides a graphical development tool (FastChip) that lets customers do the integration themselves.

In a more limited way, the Cjip's veripherals are also reminiscent of recent announcements by Altera and Xilinx. Both those companies plan to make FPGAs with hard embedded-processor cores (an ARM9 or a MIPS 4Kc for Altera, and an unnamed PowerPC core for Xilinx) that leave thousands or millions of programmable gates available for implementing application-specific coprocessors and peripherals. But the Altera and Xilinx solutions will be significantly larger, more power hungry, and more costly than the Cjip, and they are unlikely to compete for the same customers.

Multiple Instruction Sets

By far the most interesting feature of the Cjip is its ability to change instruction sets, even after booting up. The GP1000 has a proprietary Z80-like register-based instruction set, a Java-bytecode instruction set, and a stack-based instruction set optimized for C, C++, and Forth.

The new stack-based instruction set works with a new C compiler from Astrosoft, a St. Petersburg–based company headed by a former Soviet submarine captain. Astrosoft is also developing a C++ compiler that's scheduled for release in

December. These compilers typically need only one byte to encode an instruction and manipulate most operands on the top of the stack. (The Cjip can store the top 256 bytes of the stack on chip to minimize slow memory accesses.) According to Imsys, the stack-based Astrosoft compilers produce very dense code that compares favorably with Java bytecodes.

To further optimize the performance of code written in high-level languages, Imsys is developing custom assembly-level instructions that speed up C library functions and Java API routines. Like the GP1000, the Cjip has special string-handling instructions and a 256-byte on-chip string buffer. Imsys has also increased the number of internal conditions that microinstructions can test, which reduces the number of steps in critical microcode loops.

To accelerate Java performance, the Cjip now maintains two stacks: a standard JVM stack (the "evaluation stack") and a stack for local variables (the "locals stack"). As Figure 2 shows, the Cjip caches the tops of both stacks on chip in the same 1K of SRAM that holds the eight duplicate register banks.

To speed up the Java garbage collector, which periodically de-allocates memory no longer needed by objects, the Cjip uses a special garbage collector written in microcode and C. The aJile processor also implements a garbage collector in microcode, but with a further refinement: the aJ-100 can run multiple JVMs as independent processes and allow each one to observe different rules for memory management. One of those JVMs can guarantee real-time response times by disabling the garbage collector altogether, while the other JVMs operate normally. Imsys doesn't claim that embedded developers can write real-time applications in Java for the Cjip. But real-time processes written in C, C++, assembly language, or microcode can run in parallel with a Java process to provide deterministic interrupt handling.

Like aJile with the aJ-100, Imsys has implemented the JVM's thread scheduler in microcode. This should significantly improve the performance of Java software, because Java encourages multithreading, and many classes in the Java API are thread synchronized.

Compared with the GP1000, the Cjip implements a larger percentage of the Java bytecode instruction set in microcode—although still not as much as the aJ-100 does. The Cjip executes about 84% of Java's bytecode instructions in microcode, compared with about 99% for the aJ-100. As expected, the Cjip doesn't implement the most complex and rarely encountered bytecode instructions. Like other Java chips, it traps and executes those bytecodes in software. But even in some of those cases, the Cjip improves Java performance by executing the trapped bytecodes with special assembly-language instructions.

In fact, the Cjip natively executes so many low-level functions of a JVM that it doesn't need a JVM at all when it is used with Imsys's own embedded RTOS, known as Moose. Together, the Cjip and Moose provide a Java platform that complies with Sun's Java 2 Micro Edition (J2ME) for embedded systems. As Figure 3 shows, J2ME supports multiple subplatforms aimed at different segments of the embedded market.

The Imsys solution saves developers the cost of separately licensing a JVM from Sun and an RTOS from another party. In some cases, the unit-cost licenses for a J2ME-compliant JVM and RTOS might cost more than the Cjip and Moose ($19 in 10,000-unit volumes), so the Imsys package is a bargain. A similar solution is offered by aJile: the aJ-100 costs $15 in 10,000-unit volumes, is also J2ME-compliant, and doesn't need a separately licensed JVM or RTOS either.

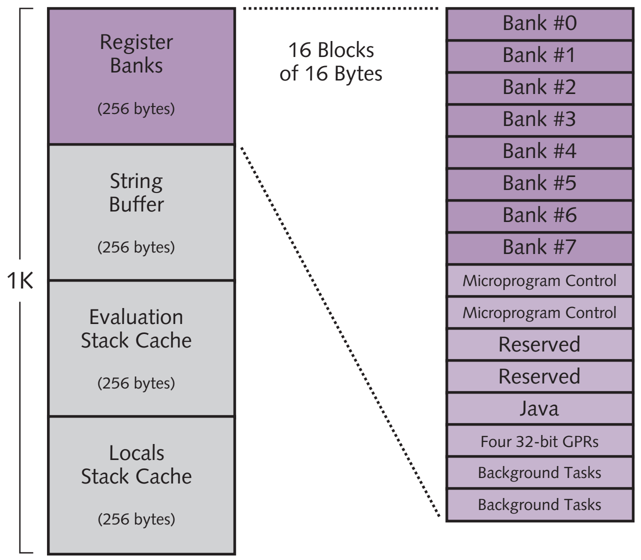

Figure 2. For faster Java execution, the Cjip stores the tops of both JVM stacks (the evaluation stack and the local-variables stack) in a pair of 256-byte caches of on-chip SRAM. Another 256-byte region of SRAM is a string buffer, and another 256-byte region stores the register banks for microcode processes.

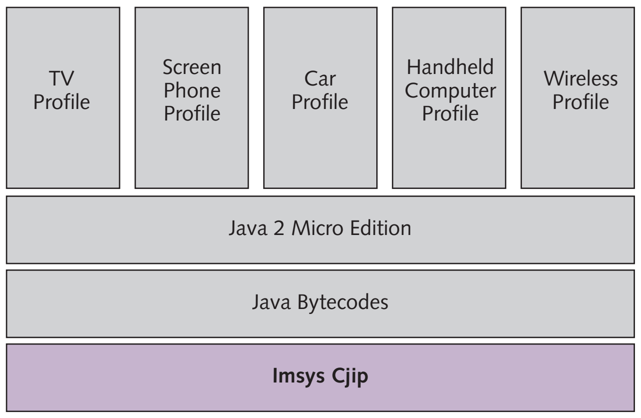

Figure 3. Java 2 Micro Edition is an embedded Java platform that supports multiple "profiles" or subplatforms, each designed for a different class of embedded applications. Imsys has an RTOS that eliminates the need for a separate J2ME-compatible JVM.

Price & Availability

Production volumes of the Imsys Cjip are scheduled to be available in September for $19 in 10,000-unit quantities. A Windows-based development kit that includes a Cjip on a PCI evaluation board, the Moose RTOS, and the Imsys Developer software (which supports Java, C, and assembly language) costs $995 and will also be available in September. For more information, go to *www.imsys.se*.

Avoiding the Java Jihad

By allowing developers to program the Cjip in C, C++, Forth, and assembly language as well as in Java, Imsys is hedging its bets for an embedded market that has yet to embrace Java. Indeed, the Cjip offers the best of both worlds. Because it allows processes written in other programming languages to run alongside processes written in Java, it can deliver real-time performance for critical control functions while giving developers the option of using Java for higher-level user-interface code. Java's rich APIs are ideal for that purpose. In the future, when the now-evolving real-time specification for Java is adopted, developers can begin using Java for the more critical functions, or they can stick with their existing code.

When judged strictly as a Java processor, the Cjip is at a disadvantage against aJile's aJ-100. The aJ-100 is probably faster by virtue of its higher clock frequency (100MHz vs. 66MHz), greater architectural width (32 bits vs. 16 bits), and more extensive microcoding of Java bytecodes (99% vs. 84%)—although the scarcity of reliable Java benchmarks makes that conclusion little more than an educated guess. Power consumption is very close: aJile claims less than 100mW for the aJ-100, but that's "typical" consumption at the chip's nominal core voltage of 2.5V. Imsys claims the Cjip's worst-case power consumption is 165mW at 3.3V or 135mW at 3V. Keep in mind that the aJ-100 is manufactured in a 0.25-micron process, while the Cjip is still at 0.35 micron. A process shrink would put the Cjip in a more advantageous position.

Both processors comply with Sun's J2ME specification, and both offer a total solution that obviates the need for a separately licensed RTOS. Only aJile promises that software written in Java can deliver deterministic, real-time interrupt handling. Imsys is partial to Java, but it isn't a soldier in the Java jihad to the same extent that aJile is. Partly for this reason, and also because Imsys had a head start, the Cjip's predecessor already has some design wins to its credit.

Neither the Cjip nor the aJ-100 would compete directly against Sun's stillborn microJava 701 (see *MPR 11/17/97-02*, "MicroJava Pushes Bytecode Performance") or other Java chips based on its picoJava core. Sun designed picoJava to run at clock frequencies upwards of 200MHz in a 0.25-micron process, and the microJava 701 typically consumes 3–4W. The Imsys and aJile processors are aimed at lower-performance, lower-power embedded systems, including battery-powered devices. Even so, one Imsys customer found the Cjip's predecessor fast enough for a color laser printer.

Patriot's PSC1000, which has been shipping for about four years, costs even less than the Cjip—under $10 in 10,000-unit volumes. It's a 32-bit processor with a 32-bit memory interface, and it has on-chip memory for Java's stacks. The core frequency is 120MHz at 3.3V or 67MHz for a 5V part. Typical power consumption is 125mW at 120MHz or 210mW at 67MHz. But unlike the Cjip or the aJ-100, the PSC1000 doesn't have a DRAM controller or as many integrated peripherals. The PSC1000 requires a companion chip known as the VPU (virtual peripheral unit) to handle some of the I/O and peripheral functions that are integrated in the Cjip and aJ-100.

Another competitor for native Java execution is JSTAR, a licensable coprocessor that translates bytecodes into native code and works with almost any CPU core (see *MPR 3/27/00-04*, "JSTAR Coprocessor Accelerates Java"). JSTAR steps out of the way when native code is executing, so it offers the same options to use other programming languages as the Cjip. And JSTAR is potentially a faster solution, because its performance scales with the frequency of the host CPU. But JSTAR is intended for ASIC integration, while the Cjip, aJ-100, and PSC1000 are standard parts. If time to market is a critical requirement, or if a customer lacks the resources to design an ASIC, an off-the-shelf solution is better.

Imsys has invested considerable effort in developing a Java personality for the Cjip (and for its predecessor, the GP1000), so obviously the company believes Java has a future in embedded systems. Others remain unconvinced. The lackluster reception for Java chips—and for past attempts to sell language-specific processors—is not a good sign. Ultimately, processors like the Cjip, aJ-100, and PSC1000 must compete against dozens of embedded processors that may have higher performance, lower power consumption, greater integration, or lower cost. Embedded developers must decide if those advantages are worth sacrificing for the claims that Java is a safer, more productive, and more portable programming language. Although we think those claims have a great deal of merit, we also like the strategy favored by Imsys—offering developers the options of using Java exclusively, sparingly, or not at all. ◇