GP1000 Has Rewritable Microcode

Imsys Processor Executes Java Bytecodes and Concurrent Microcode Processes

by Tom R. Halfhill

Never mind RISC and CISC. "NISC" and "WISC" are some of the fanciful terms suggested for a unique embedded processor from Sweden that has rewritable microcode, microcode-level concurrency, native Java bytecode execution, multiple register banks, and other unusual features.

Imsys AB (Stockholm) has designed the new chip, dubbed the GP1000, for two roles. Conservatively, it's a low-cost (under $25), low-power (under 300 mW at 33 MHz) controller for multifunction peripherals—the combination printer/scanner/fax/copier devices that are popular in home offices and other crowded workplaces. For that application, the GP1000 provides concurrent microcoded processes that support scanning, image buffering, compression/decompression, halftone screening, printing, and other imaging tasks that normally would require high-level programming.

More ambitiously, Imsys has spun off subsidiary Clean Bean to market the CPU's Java capabilities. By programming the microcode to execute Java bytecodes and some higher-level functions of a Java virtual machine (JVM), Imsys has created one of the few Java chips not derived from Sun's PicoJava. In this role, the GP1000 is aimed at smart appliances that can benefit from Java's network and platform portability: handheld computers, multifunction cell phones, point-of-sale terminals, and other intelligent devices.

The rewritable microcode allows Imsys to customize the GP1000 for a wide range of tasks—essentially creating new instruction sets for almost any purpose. For example, a network router could do table lookups and checksum calculations in a microcode process, or a custom assembly-language instruction could speed up an application-specific loop. It's like inline assembly optimizing in C, except one level deeper.

One GP1000 customer refers to this chameleon-like capability as NISC (no instruction set computing). Imsys engineers aren't too fond of that term, though. They're a little more amenable to WISC (writable instruction set computing), which was the basis of a theoretical processor described in 1987 by maverick CPU architect Phil Koopman.

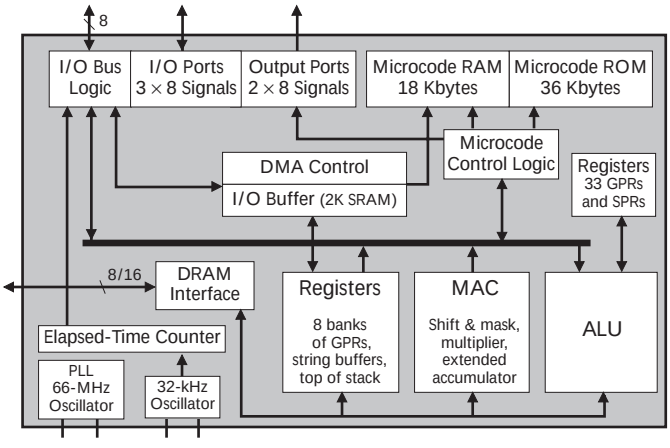

Figure 1. The GP1000 is a scalar embedded processor with large blocks of SRAM and ROM to store its programmable microcode. The MAC unit is accessible only with microinstructions.

It's Really CISC—With Twists

At its core, the GP1000 is a CISC architecture that inherits its designers' experience with document processing and virtual machines. Its earliest ancestor was a three-board TTL processor in the early 1980s that had a microcoded virtual machine for the UCSD p-System. A later implementation took the form of a gate-array chip for desktop publishing terminals.

As Figure 1 shows, the GP1000 is a scalar CPU with a single ALU, a MAC unit (accessible only from microcode), multiple register banks, an 8/16-bit memory bus, and a separate 8-bit data I/O bus with four DMA channels. The I/O bus runs at clock ratios of 1:1, 1:2, or 1:4 relative to the core frequency, at bus speeds up to 33 MHz. Current samples of the GP1000 are running at a core speed of 33 MHz. The engineers believe they can push the chip to 50 MHz in the current process with a minor tweak of the microcode timing.

There are also 24 I/O pins and 16 output pins for serial signals. A microcode process can read or write a group of eight pins at the same clock frequency as the I/O bus.

The data bus requires a small amount of external glue logic, currently implemented in an FPGA. The type of FPGA depends on the I/O requirements. On one sample board, Imsys uses a small device that costs about $3; another board has a more expensive 208-pin FPGA with a PCI interface. Even the small one can handle all four DMA channels, yielding a total throughput of 33 Mbytes/s.

LCD panels are pretty standard on multifunction peripherals and the GP1000 can support virtually any kind of LCD via microcode routines and its programmable output ports. If the LCD has a frame buffer, the GP1000 writes to it only when the image changes, so the serial ports are fast enough for that purpose. If the LCD doesn't have a frame buffer, the higher demands for refreshing the image require DMA transfers over the data bus. There's enough bandwidth to drive a 640 × 480-pixel color LCD—a more luxurious screen than a peripheral is likely to have. A more reasonable 128 × 64-pixel monochrome LCD consumes only 9% of the I/O bus's DMA capacity.

The GP1000 is available directly from Imsys/Clean Bean. Imsys's manufacturing partner (and one of the first

customers for the chip) is Stockholm-based Ericsson Components. Ericsson contributed a great deal of expertise toward shrinking the design to fit a 3.3-V, 0.35-micron, three-layer-metal CMOS process. The fab is UMC in Taiwan.

At 0.35 micron, the GP1000's die measures a diminutive 16 mm^2—with only about 20% of that area consumed by the core logic, as Figure 2 shows. On-chip SRAM and ROM, filled mostly with microcode, account for the remainder of the 1.4-million-transistor die. It's packaged in a 144-pin low-profile quad flat pack (LQFP) that's 1.6 mm high.

Imsys designed the logic elements to be easily separable from the on-chip memory if customers have a more specialized application in mind. The core has approximately 70,000 logic transistors. A synthesizable VHDL version is scheduled to be finished by March, and a process shrink to 0.25 micron should follow shortly afterward. That shrink would also enable higher integration. Stefan Blixt, chief architect of the GP1000, thinks he can add 1.5M of DRAM to the processor while holding the die to 36 mm^2 or less at 0.25 micron.

Triple-Threat Multitasking Is Unusual

One of the most interesting features of the GP1000 is its multiple levels of concurrency. Potentially, there are three: concurrent microcode processes, assembly-language multitasking, and Java multithreading. Rewritable microcode is the foundation of this unusual architecture.

The GP1000 stores most of its microcode in 36 Kbytes of on-chip ROM and the rest in 18 Kbytes of on-chip SRAM. Therefore, two-thirds of the microcode is static and one-third is rewritable. The microcode ROM and SRAM are organized in 72-bit words to match the length of the microinstructions. The microcode cycle time is 30 ns at a PLL-oscillator frequency of 66 MHz. Most of the microinstructions execute in two cycles while a few require three. An assembly-level instruction requires at least two microinstructions.

At startup, a boot loader in the processor's ROM loads the rewritable portion of the microcode from an off-chip source, such as EPROM or flash ROM. After the system boots and initializes memory, the processor can load new microcode from system DRAM. This sequence allows customized or patched microcode to reside in EPROM, flash ROM, or DRAM. After startup, a program can use a special assembly-language instruction to load fresh microcode at any time—an intriguing (and potentially hair-raising) feature that allows applications to alter the CPU's instruction set on the fly.

A control microprogram written entirely in microinstructions handles concurrency at the microcode level. It's almost a tiny operating system. It manages interrupts, I/O, and process scheduling for microcode processes. And those processes are a far cry from the atomic assembly-language instructions that are normally the *raison d'être* for microcode. In the GP1000, a microcode process can perform such high-level image-processing tasks as halftone screening, adaptive thresholding, and RGB-to-CMYK colorspace conversions. For example, one microcoded assembly instruction designed for use with multifunction peripherals can apply CCITT compression to an entire scan line of image data at once.

In effect, the application software drives the instruction set of this processor, not the other way around. One common use of this feature will be to compress application-specific loops. An assembly instruction sequence such as MOV-CMP-BEQ (move to register, compare, and branch if equal) can collapse into a single assembly instruction called MCB. Combining instructions in this way doesn't change the total execution latency, but it saves overhead, because the CPU doesn't have to fetch as many instructions; off-chip references are especially costly for a small processor with an 8-bit bus.

An example of the efficiency of this approach is an optimized assembly instruction that accelerates error-diffusion processing. A loop in this algorithm that required 23 instructions per iteration on a Pentium was reduced to a GP1000 instruction that needed only 8.25 microinstructions (eight per loop plus one every fourth lap). The GP1000 essentially matched the performance of a 133-MHz Pentium, even though the Pentium is a much more powerful processor.

The GP1000's malleable instructions wouldn't be possible in a RISC architecture that forgoes microcode in favor of hard-wired, fixed-length instructions with finely tuned latencies. Assembly instructions on the GP1000 can have arbitrary lengths and latencies. Normally, the disadvantages of such an extreme CISC philosophy would be difficult instruction decoding and the nightmarish task of synchronizing parallel execution pipelines. But the GP1000's fixed-length microinstructions simplify the decoding somewhat, and its scalar core doesn't have multiple pipelines to worry about.

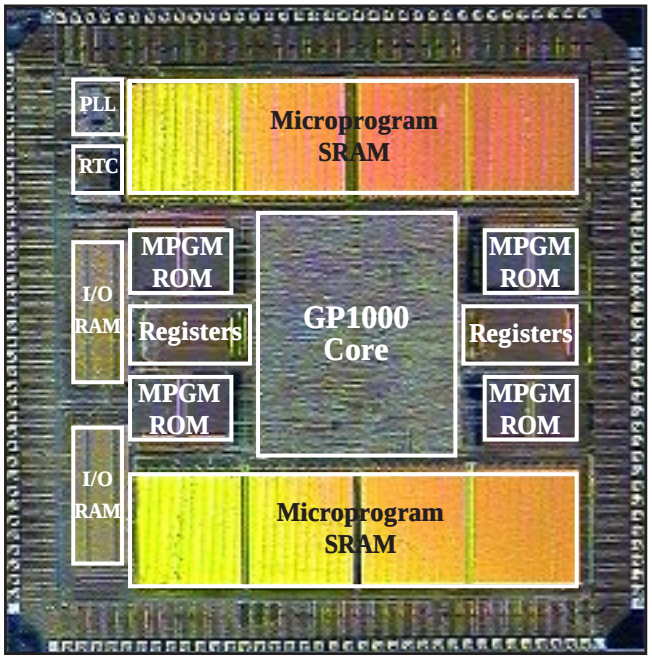

Figure 2. Initial GP1000 chips are manufactured on a 3.3-V 0.35-micron process that fits 1.4 million transistors on a 4 × 4-mm die. The core logic accounts for only about 20% of that area.

Price & Availability

The GP1000 is currently sampling at 33 MHz (66-MHz oscillator), and production shipments are scheduled for 1Q99. Its list price will be $25 in 10,000-unit quantities. A development kit will also be available in 1Q99 for $995. For more information, access *www.javamachine.com*.

Microcode programming isn't something that Imsys is dumping on customers. It will be done in house to customer specifications, using a visual development tool that Imsys created for that purpose. (It's an extension of the visual assembler that Imsys supplies to customers.) Imsys doesn't think many customers will be eager to tackle microcode programming anyway. Even a veteran assembly-language coder might blanch at the 72-bit-long microinstructions—each one has about two dozen fields and can jump to another address.

Java Was an Afterthought

Development was already well under way when Imsys engineer Roger Sundman wondered if the microcode could handle Java bytecode. Compared to image processing, Java is almost easy. Imsys engineers have already implemented most of the 226 Java bytecode instructions in microcode for the GP1000, but they will probably stop short of implementing all of them. (In comparison, Sun's MicroJava 701 implements about 170 bytecode instructions in hard-wired logic, executes about 30 in microcode, and traps the remaining bytecodes in software.) Some instructions—such as the one that creates a new object—are exceedingly complex and don't occur often enough in programs to warrant native support.

Most of the Java-specific microcode loads into the GP1000's on-chip SRAM at bootup, but the bytecode decoding table is stashed in on-chip ROM. Imsys is also implementing some higher-level Java functions in microcode. For example, the JVM's garbage collector is a mark-and-sweep algorithm that is notorious for consuming CPU cycles at the most inopportune moments. Imsys engineers were hard at work this fall rewriting the garbage collector in microcode.

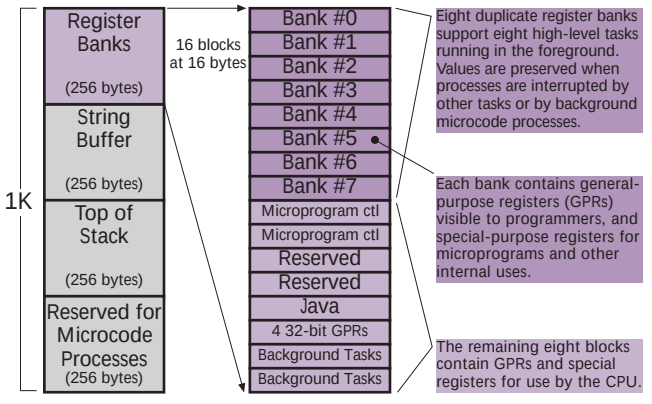

Figure 3. Eight register banks allow the GP1000 to switch among concurrent processes without saving or restoring registers.

The type of JVM can vary according to the application. For multifunction peripherals, Imsys says Embedded Java (see MPR 4/20/98 p. 8) is best. Personal Java is a possibility for consumer-electronics devices. After a process shrink to 0.25 micron, the GP1000 might be suitable for Java Card, a JVM that meets the ISO 7816-4 standard for smart cards.

Imsys claims that the advantages of native Java support are faster application development, easier networking, and consumer-friendly field upgrades. High-level applications for intelligent devices that have communications capabilities (such as smart cell phones and multifunction peripherals) are certainly easier to write in Java than in assembly language or C/C++. Such devices could use Java's networking capabilities to download software patches and upgrades with little or no user intervention. And native Java execution conserves memory that otherwise would be occupied by a Java bytecode interpreter and a just-in-time (JIT) compiler.

Of course, an embedded processor that avoids Java altogether could conserve even more memory. The GP1000's success as a Java chip will largely depend on the popularity of Sun's Personal Java and Embedded Java platforms. The outlook for those Java subplatforms is uncertain—they're even newer and less mature than their parent platform.

But Java isn't the only programming option. In addition to the assembler that Imsys provides with the development kit, there's also a C++ tool that compiles straight to Java bytecodes. This option is for customers who aren't enamored with Java or must port existing code to the chip. In an interesting twist, Imsys subcontracted the development of the C++ compiler to a Russian company in St. Petersburg that's run by a former Soviet submarine captain.

Multiple Register Banks Aid Context Switching

Another unusual feature of the GP1000 is its register file. It has eight duplicate banks of registers for context switching between concurrent processes.

As Figure 3 shows, most of the registers reside in 1K of on-chip memory, which is divided into four 256-byte regions. (33 GPRs and several special registers reside elsewhere on the chip.) The GP1000 uses one of the 256-byte regions as a buffer for complex string instructions, another for the Java stack (or any other stack, for that matter), and a third for microprograms. The remaining 256-byte region is subdivided into 16 blocks, each containing 16 bytes. Half of those blocks store the eight banks of GPRs; the other blocks contain special registers not visible to programmers.

Each bank has seven 8-bit GPRs (with 8080/Z80 mnemonics) that programmers may combine into three 16-bit GPRs. These are the only registers visible to high-level programs. The banks support up to eight foreground processes that can interrupt each other or be interrupted by microcode processes without incurring the overhead of dumping the registers to memory and restoring them later.

The other registers in each bank are invisible to high-level programs. They serve various internal purposes for microcode processes, native Java bytecode execution, and so forth. Imsys reserves some spaces for future use.

The GP1000's MAC unit, like many of its registers, is not directly accessible from high-level software. It's available to microcode processes, which can execute an 8 × 8-bit multiply-add operation plus a multiway jump in a single microinstruction. The MAC also comes into play when standard assembly instructions or bytecode instructions perform multiplication. The MAC's accumulator is 18 bits wide to store the extended results of iterative operations.

Power Consumption Fits Mobile Requirements

Imsys has been testing the first samples of the GP1000 since July. According to architect Blixt, the part consumes less than 400 mW at 33 MHz when executing a worst-case loop that exercises the ALU, MAC, string buffer, and some registers during each cycle. A typical program consumes 300 mW or less.

Although Imsys has found no bugs, the engineers have identified a shortcoming that was not apparent during the early design stages. The microcode ROM and SRAM both draw power at the same time, even though only one is accessed each cycle. Alternating power between the memories as needed would be more efficient. Blixt thinks this could cut power consumption by 35% to 40%; the memories now consume about 75% of the chip's power.

Such a change might not have to wait until the next full revision. Imsys has halted some partially finished wafers in the fab and is preserving them in nitrogen gas, hoping it can implement the change in the remaining masks. If that's possible, the power consumption for a typical program could drop to around 200 mW at 33 MHz in the 0.35-micron process. More savings would accompany a shrink to 0.25 micron.

Power consumption is already well below the requirements of line-powered devices, such as multifunction peripherals. Imsys's goal, however, is to reduce the power envelope so the GP1000 will be suitable for a wider range of battery-powered mobile devices.

Difficult to Categorize

Rewritable microcode isn't a new idea. Neither is microcode support for high-level languages. Digital's LSI-11 had a writable control store, and Lawrence Livermore Laboratory implemented UCSD Pascal in LSI-11 microcode. The Burroughs B1700 had microcoded instruction sets for BASIC, FORTRAN, COBOL, and RPG-II. In the 1980s, LISP machines from Texas Instruments and Symbolics had microcoded garbage collectors. Even Intel's P6-series processors have a tiny amount of rewritable microcode to allow patches.

But among current processors, the GP1000's combination of rewritable microcode, native Java execution, microcode-level concurrency, microcode support for image processing, and multiple register banks is unique. Those attributes also make it difficult to compare the GP1000 with competing products. Clearly, it's more than a run-of-the-mill controller for multifunction peripherals. Even among Java chips, the GP1000 is an oddball. It wasn't originally designed as a Java chip, it owes nothing to Sun's widely licensed picoJava core, and it can change identities literally in a flash.

Feature	Imsys GP1000	Sun MicroJava 701
CPU Frequency	33 MHz	133–200 MHz
Bus Frequency	33 MHz	50–100 MHz (mem), 33/66 MHZ (PCI)
Clock Ratios	1:1, 1:2, 1:4	1:2, 1:3, 1:4
Data Bus	8 bits, 4 DMA ch	8/16/32-bit I/O, 32-bit PCI
Memory Bus	8/16 bits, to 128M	32/64 bits, to 256M
Transistors (total)	1.4 million	4.1 million
Transistors (core)	70,000	n/a
µCode ROM/RAM	36/18 Kbytes	n/a
Special Features	Rewritable µcode; µcode concurrency; Native Java ex; 8 register banks	Native Java ex; Integrated PCI ctl; Memory controller; Floating-point unit
Process Technology	0.35-micron CMOS	0.25-micron CMOS
Voltage	3.3 V	2.5 V (3.3-V I/O)
Die Size	16 mm^2	67 mm^2
Packaging	144-pin LQFP	316-pin BGA
Power Dissipation	<300 mW @ 33 MHz	<4 W @ 200 MHz
Availability	1Q99	1Q99

Table 1. The GP1000 is aimed at multifunction peripherals and mobile Java devices. Sun's more powerful 701 is primarily for similar appliances. n/a = information not available (Source: vendors)

Sun's MicroJava 701 (see MPR 11/17/97, p. 9) is more like a mainstream desktop processor in terms of resources. As Table 1 shows, the 701 easily outclasses the GP1000 in that regard, but it also has a much larger die on a smaller process and dissipates ten times as much power. Without benchmarks, it's hard to compare these widely disparate Java chips, but they're obviously designed for different applications. The power requirements alone make the GP1000 more suitable for mobile devices.

The GP1000 also is comparable to Patriot Scientific's PSC1000 (see MPR 4/15/96 p. 1), another Java chip that began as something else (a stack-oriented Forth chip). The PSC1000 is more of a general-purpose microcontroller and sells for under $10. Its approach to Java is quite different: a custom bytecode verifier maps bytecodes to native CPU instructions. And it lacks the GP1000's microcode-level concurrency and image-processing features.

Until the fog clears over the unproven market for Java chips, the GP1000's future in that role is uncertain. Fortunately, Imsys hasn't lost sight of its original goal of designing a versatile CPU for multifunction peripherals. Although Imsys is a small company focused on development, Ericsson and another unnamed business partner will help Imsys/Clean Bean market the GP1000 as both a Java chip and a peripheral controller. We will see how many customers take advantage of this processor's unique capabilities. M

Tom R. Halfhill has been a technology writer since 1982 and was formerly a senior editor at BYTE Magazine. *Tom can be contacted at* halfhill@hooked.net.