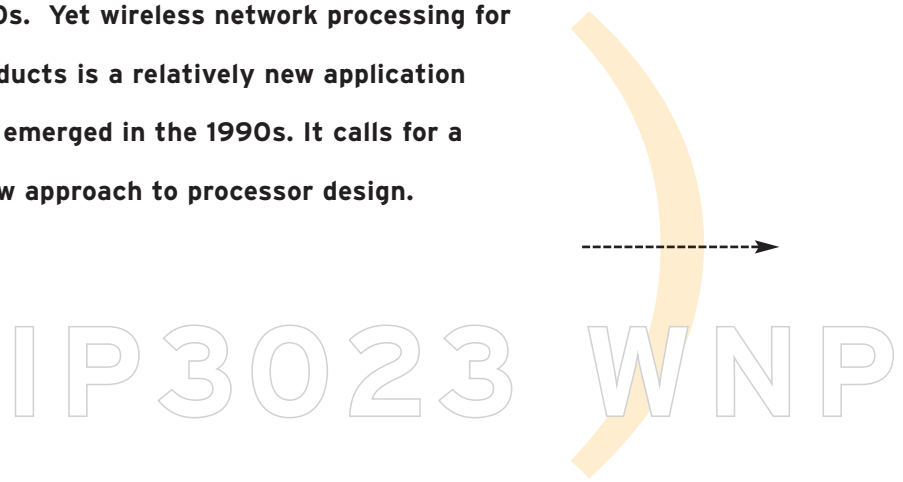
The Ubicom IP3023™ Wireless Network Processor

White Paper

A Next-Generation Packet Processor for Wireless Networking

The Internet has been around in one form or another since the 1960s. Yet wireless network processing for embedded products is a relatively new application category that emerged in the 1990s. It calls for a completely new approach to processor design.



UBICOM™

635 Clyde Avenue
Mountain View, CA 94043
tel 650.210.1500
fax 650.210.8896
www.ubicom.com

April 15, 2003

Contents

I. Introduction 3

II. The Challenge of the Wireless Network 3

III. The MASI Architecture 6

IV. Real-Time Multithreading 9

V. An Optimized Instruction Set 12

VI. The Software I/O Advantage 14

VII. Conclusions 19

I. INTRODUCTION

Wireless networking is a fundamentally different challenge for microprocessors than desktop, server, or embedded microcontroller applications. The nature of the flow of data through the network, the stringent requirements on responsiveness and throughput, the rapid change of protocol standards and the need for low-cost solutions create a difficult problem for today's microprocessors to solve. In addition, testability, ease of use for end-users and ease of design-in are requirements for most devices that use wireless networking.

A new type of processor is needed. This processor must efficiently handle large bursts of transient data, sustain rated throughput, be very flexible and simple to use in hardware and software, and offer a low-cost system solution to designers. This paper will discuss the first such processor – the Ubicom IP3023 wireless network processor.

The development of the Ubicom IP3023 has led to the definition of a new processor architecture, beyond CISC and RISC. This new architecture type, Multithreaded Architecture for Software I/O (MASI), uniquely meets the requirements of the wireless network and offers several other advantages as well. Ubicom's IP3023 wireless network processor, based on MASI, enables new wireless applications through several features, including:

- Eight-Way, Deterministic Hardware Multithreading, which provides high sustained throughput and very efficient operation of multiple tasks simultaneously;
- Zero-Cycle Context Switching for very fast interrupt handling;
- Dual-Operand Memory-to-Memory Addressing Modes that enable fast operations on packet data, and small code size;
- Simple, Clean Instruction Set Architecture that is designed specifically for network traffic; and
- Software I/O, which executes all I/O operations and protocols in software, offering very flexible system design, and eliminating most of the I/O hardware.

We will examine each of these features in detail.

II. THE CHALLENGE OF THE WIRELESS NETWORK

PCs are the most visible clients on the Internet today. Indeed, most people view the Internet as nothing more than a worldwide network of PCs. But that's changing fast. Before long, the few hundred million PCs will account for a tiny minority of Internet-connected clients. They will be far outnumbered by Internet-enabled phones, hand-held computers, and miscellaneous embedded devices. The miscellaneous category encompasses everything from industrial machines and motor vehicles to traffic lights, toys, appliances, heating/cooling systems, MP3 players, video game consoles, cellular base stations, broadband modems, home Internet gateways, and almost anything else that can benefit from an Internet connection. This vision of a cloud of interconnected devices is known as "Ubiquitous Communication."

When there are billions of Internet clients, most of the Internet will seem invisible to the average person. The typical client will no longer be a conventional computer with a screen and a keyboard. But no matter what form they take, all Internet clients will always have two things in common – an Internet connection (either continuous or intermittent), and the ability to process the network packets they receive through that connection.

Wireless networking will spark mass sales of these interconected devices. The transition to ubiquitous communication is beginning now, because sophisticated, inexpensive, easy-to-configure products that support wireless protocols such as Bluetooth® and 802.11 are in stores now.

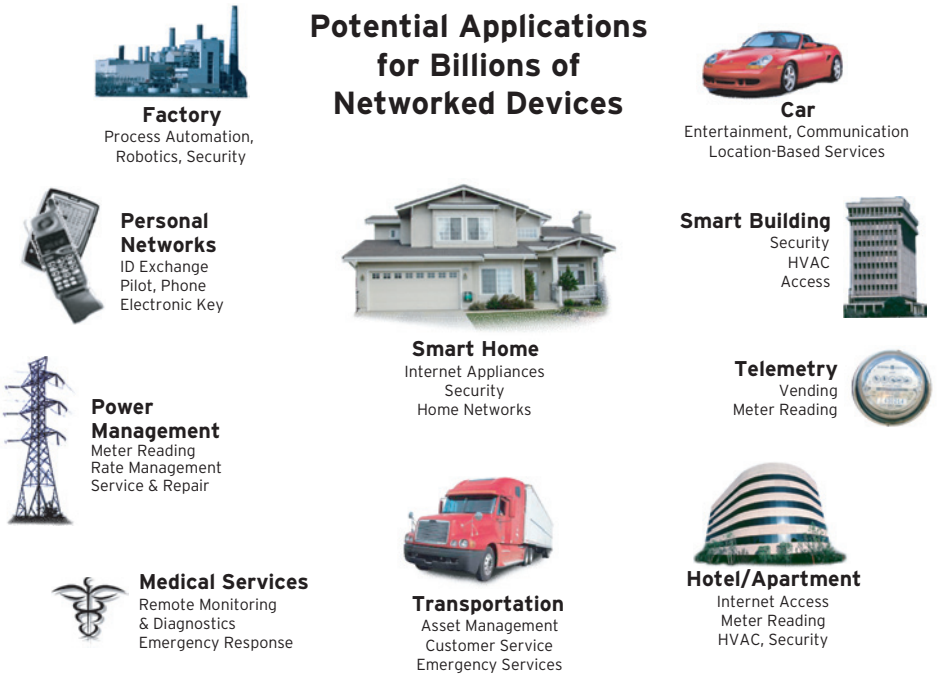


Figure 1: Embedded network connectivity is redefining Internet clients.

These new protocols are driving development of new technologies. The product requirements in wireless networking are different than legacy wired environments, and packet processing requirements are significantly different than traditional data processing:

- Low-latency packet processing is as important as high throughput.
- There are more communication protocols to accommodate.
- Cost is often a crucial factor, especially for consumer products.

Consider the first issue: packet processing. This vital job usually falls to a specialized processor that bridges the flow of packets from the Internet to a general-purpose CPU elsewhere in the device. At the Internet core, high throughput is paramount, because the routers and servers are large, complex machines that keep the network humming by managing the flow of billions of packets toward their destinations. The relentless demand for high performance justifies costly investments in the very fastest equipment.

A wireless network processor is a different kind of processor for wireless networking. It is needed to handle only the packets destined for the client it inhabits or perhaps a few other clients to which it's attached. This processor rarely deals with actual connection speeds exceeding 100 megabits per second (Mbps). Therefore, high throughput is less important than low latency. In this context, *latency* means the amount of time the wireless network processor spends on verifying and analyzing each packet. The processor must handle each arriving packet quickly to avoid dropping the next one, because most wireless network products probably can't tolerate the high cost of memory to buffer large numbers of incoming packets.

Another essential issue in the design of processors is which communication protocol to support. Core routers usually focus on a single Layer 2 protocol. Wireless networking products, on the other hand, must deal with a wide range of protocols. The wireless protocols themselves, typically one or more of 802.11a, 802.11b and 802.11g (also known collectively as Wi-Fi®), and Bluetooth are required for all wireless products. In addition, access points, bridges and gateways also need to support and convert wired protocols such as Ethernet and TCP/IP, HomePlug®, USB, ADSL, DOCSIS (for cable modems), PCI, PC Card (PCMCIA), ISDN, and GPRS (General Packet Radio Service). Moreover, these products must be able to adapt to the continuing evolution of protocols, as new features and capabilities are added after standards are ratified.

This creates a dilemma for the designers of wireless network processors – and for their customers who develop wireless products. Supporting every conceivable protocol in silicon on one chip would vastly inflate the size and manufacturing cost of a wireless network processor. Supporting only one or two protocols while excluding others would be a long-shot gamble that only psychics are likely to win. Supporting many different protocols on many variations of the same chip would complicate inventories and make choices more difficult for product developers.

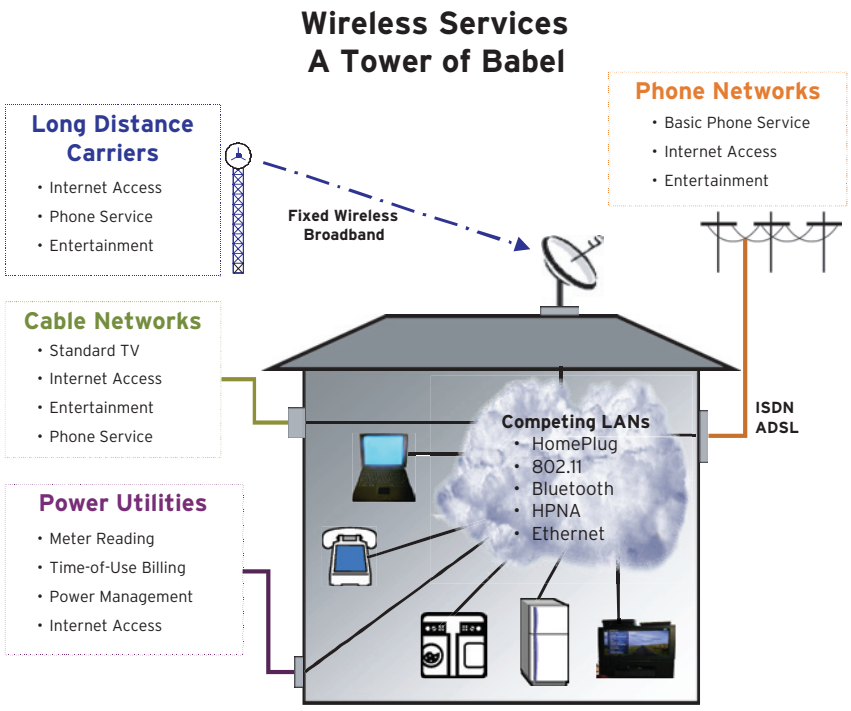


Figure 2: Wireless protocols are like a Tower of Babel.

The third significant factor that affects the design of wireless network processors is cost. Unlike the expensive servers and routers that take care of business deep inside the Internet, wireless devices are often affordable consumer products, such as MP3 players, broadband modems, and telephones. Most of them must run on batteries, which imposes an additional requirement for low power consumption. Indeed, the penetration of Internet connectivity in this market depends almost entirely on driving down the implementation cost. As the incremental cost of connectivity falls, more everyday products will become Internet clients. Cutting costs requires a low bill of materials (BOM) and efficient power consumption – without sacrificing the low-latency performance and support for multiple protocols that are also vital for success.

III. THE MASI ARCHITECTURE

Microprocessor architectures always reflect the applications and technical constraints that are dominant when the architecture is created. For instance, when DRAM was exorbitantly expensive in the 1970s, engineers designed CISC architectures with compact variable-length instructions to conserve memory. In the 1980s, DRAM prices plunged, more programmers began writing software with compilers instead of assemblers, and CPU clock frequencies soared. Engineers responded by creating RISC architectures with fixed-length instructions for easier decoding, larger register files for easier compiling, and register-to-register instructions that made fewer accesses to memory – which wasn't keeping pace with CPU clock frequencies. In the 1990s, the growing popularity of multimedia and networking applications drove engineers to add instruction extensions to their architectures to address those new demands.

Unfortunately, the results of this lengthy evolution aren't entirely good. Today's most popular CPU architectures were created for entirely different products, such as PCs, workstations, servers, or non-networked embedded systems. They carry the old baggage of features designed to overcome technical constraints that may no longer exist or be important. And their growing complexity often makes them unwieldy.

As Figure 3 shows, different applications and configurations require different types of processors to operate efficiently. CISC processors such as the Intel® Pentium® are tuned to desktop applications, operating systems and data sets, while embedded devices typically use RISC processors such as those sold by MIPS® and ARM® for control and administration functions. These RISC processors are often sold as "soft" cores, bundled with other functions to create a special purpose "system on a chip" (SOC).

	Product Type	Appropriate Architecture	Product
Desktop Computers	CPU	CISC	Pentium
Servers	MPU	Multiple RISC	PowerPC®, SPARC®
Embedded	SOC	Soft RISC Cores	MIPS, ARM
Wireless Network	Wireless Network Processor	MASI	Ubicom

Figure 3: Different applications demand different processor architectures.

That's why Ubicom started with a blank slate and created a new kind of architecture optimized for the special demands of wireless network processing – the Multithreaded Architecture for Software I/O (MASI). The MASI architecture is uniquely suited for wireless network processing. Ubicom has pioneered the MASI architecture, which first debuted on the company's IP2000™ family wireless network processors, and has now been fully implemented on the new IP3023 wireless network processor. It has several interesting features:

- A small instruction set optimized for packet processing
- Memory-to-memory instructions and a dual-memory pipeline
- On-chip program and data memory, but no instruction or data caches
- Instruction-level hardware multithreading
- Fast context switching (or in the case of the IP3023, zero-cycle context switching).
- Multiple register banks for preserving thread states
- Continuous interrupt handling
- Static branch prediction with compiler hints
- No TLBs, MMUs, or other mechanisms that could impair real-time determinism

Yet the IP3023 architecture doesn't ignore the lessons learned during the past three decades of microprocessor design. Fundamentally, it shares some design principles with a traditional 32-bit RISC processor. It has a Harvard architecture, fixed-length 32-bit instructions, a RISC-like pipeline, and single-cycle throughput. From there it diverges.

Programmers are pleasantly surprised when they first open a Ubicom reference manual. The instruction set is refreshingly small. While most other CPUs have hundreds of instructions because of their PC/workstation/server heritage, the IP3023 needs only 41 instructions. The architecture is optimized for wireless network processing, not for running databases, compilers, word processors, spreadsheets, or games. As we'll describe in detail later, Ubicom carefully designed those 41 instructions to process packets more efficiently than RISC or CISC architectures. By doing the same work with fewer instructions, MASI reduces the code size of wireless network software and reduces or eliminates the need for external flash memory.

One of the most significant ways in which MASI diverges from the RISC religion is its *memory-to-memory architecture.* Many MASI instructions access memory *twice* – once to load a value from memory, and again to store the value after manipulating it. RISC architectures shun multiple memory accesses because off-chip memory latencies haven't kept pace with the rising core frequencies of CPUs. Instead, they use separate instructions to load a value from memory into a register, store the value in a register after manipulating it, and then copy the register back to memory. That still requires two memory accesses, but separating the load/store instructions allows a program to perform multiple register-to-register operations on a value before the final store. Some CPUs can rearrange the instructions to hide the latency of a load or store.

The so-called *load/store architecture* of RISC is well-suited for the software applications that RISC processors were designed to run in the 1980s and 1990s. It's not as suitable for 21st-century wireless network processing. In fact, about 40 percent of the instructions in a typical RISC instruction set are register-oriented load/store instructions. That's why the MASI architecture takes a different approach.

To begin with, a packet processor rarely needs to perform multiple operations on a fragment of packet data between loading it from memory and storing it back to memory. Therefore, it's redundant to use separate instructions that load the packet data into a register, manipulate the data in a register-to-register fashion, and then store the data back to memory. Typical packet-processing operations touch the data only once. For instance, the IP3023 wireless network processor might perform a cyclic redundancy check (CRC) to verify the accuracy of every byte in a packet or scan the packet header for a destination address. Why use multiple instructions that waste CPU cycles and inflate code size when a single instruction can do the job?

That's exactly how the MASI architecture works. A single optimized instruction can load some packet data from memory, perform the necessary operations on the data, and then store the data directly back to memory – without stopping at a register along the way. For flexibility, the memory-to-memory instructions have multiple addressing modes for base + index, base + offset, and auto-increment memory addressing.

To make this memory-to-memory architecture work, the architecture also uses a modified pipeline and fast local memory. The pipeline resembles a conventional RISC pipeline with fetch, decode, execute, and writeback stages, except it's superpipelined for higher performance. (Superpipelining is a common technique that divides single-stage operations into multiple stages to preserve single-cycle throughput and enable higher clock frequencies.)

As Figure 4 shows, the IP3023 wireless network processor divides instruction fetching and writeback into two stages. There are also a few extra stages to implement the memory-to-memory architecture. One new stage precedes the first fetch stage to manage the instruction-level hardware multithreading (more on this later). After instruction decoding, three additional stages allow an instruction to access memory before execution: one stage to calculate the address and two stages to read memory. Like the multiple fetch and writeback stages, the divided address-calculation and memory-access stages preserve single-cycle throughput and permit higher clock frequencies. After accessing memory, the instruction can perform its arithmetic or logical operation on the data during the usual execute stage. Finally, the instruction stores the result back to memory during the two writeback stages.

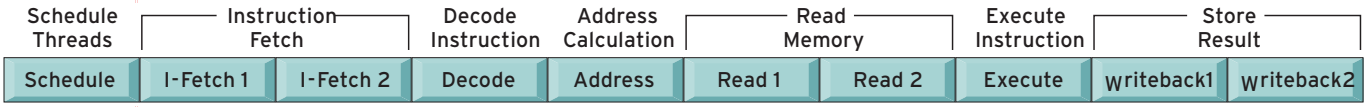


Figure 4: The IP3023 architecture has a memory-to-memory superpipeline.

This ten-stage superpipeline is deeper than a minimal RISC pipeline but still supports the single-cycle throughput that's characteristic of modern RISC architectures. One potential drawback of superpipelining is a greater penalty for mispredicting a conditional branch, because the processor has to flush and re-prime the pipeline with new instructions fetched from the correct branch address. Superpipelined RISC processors and the Intel Pentium use large, complex, dynamic branch prediction hardware to avoid some of this penalty. In Ubicom's multithreaded architecture, a mispredicted branch only flushes the instructions from the thread containing the branch. Instructions from other threads are unaffected. If there are four or more active threads, the mispredicted branch penalty is only one clock rather than six clocks in a superpipelined RISC, or 19 clocks for Pentium 4. Because Ubicom's branch mispredict penalty is so low, the IP3023 uses static branch prediction with compiler hints instead of dynamic branch prediction. It actually works better than a shorter pipeline using dynamic prediction.

Delay slots after branch instructions in the IP3023 processor typically will not affect performance due to the multithreaded architecture. Since at any given time instructions can be executed from any active thread, delay slots are ignored and instructions from another thread are executed.

In contrast, RISC machines stop and wait for the delay slot after the branch. The delays help hide the time taken to perform the branch. Converting an existing RISC architecture to operate without traditional branching is nearly impossible.

Almost any other processor would pay another performance penalty for accessing memory twice with a single instruction: the penalty of fetching data from off-chip memory after a cache miss. The IP3023 eliminates that penalty by storing packet data in fast local memory and dispensing with instruction/data caches altogether.

Instead of caches, which can impair determinism in real-time applications, the IP3023 wireless network processor has 256 KB of on-chip SRAM for program memory (P-RAM) and 64 KB of on-chip SRAM for data. The processor can access either the P-RAM or the data memory in a single cycle. This highly efficient Harvard architecture allows MASI-based processors to keep program code and data in separate on-chip memories to minimize bus conflicts.

MASI-based processors can handle most wireless applications without any external memory. The Ubicom OS and a typical protocol stack require less than 100 KB of code, leaving more than 150 KB of internal P-RAM for the application software. That keeps costs down, saves power, and reduces the size of system boards. For larger applications, the IP3023 can address up to 4 MB of external flash memory and up to 64 MB of external SDRAM. At bootup, the system can copy code or data from external flash memory to internal memory for faster access, decompressing the code or data in software if necessary.

One thing you won't find in the architecture are large FIFO buffers. That's because it doesn't need to buffer large numbers of packets or keep multiple copies of packets in various places. MASI-based processors are fast enough at typical wireless speeds to handle each packet as it arrives. Competing processors often waste time and memory by shuffling packets in and out of registers, to and from memory, and through large FIFO buffers.

Because the processors can do the same work with less internal memory, it slashes power consumption and silicon cost. Yet no matter how bursty the traffic, the processors maintain a smooth, continuous flow of packets from the Internet connection to the CPU.

IV. REAL-TIME MULTITHREADING

Another significant feature of the architecture is instruction-level hardware multithreading. Note that this doesn't require a heavyweight multithreaded or multitasking RTOS. Implementing multithreading in hardware at the instruction level allows the processor to handle interrupts with incredible speed while shrinking the size and increasing the reliability of the Ubicom ipOS™ operating system. Instruction-level multithreading simplifies the programming model, too, because it's managed by the processor.

Almost all other microprocessors rely on the OS to manage multithreading. Those CPUs can't handle more than one thread context in their pipelines at the same time. Switching contexts requires them to stall the pipeline, copy all their registers and other machine states to memory, and then restart the pipeline by fetching instructions from the new context. This, in turn, almost always forces the processor to flush its caches, because the new instructions are probably in another region of memory. The whole procedure wastes hundreds of CPU cycles. It also impairs the processor's ability to respond deterministically to interrupts.

In stark contrast, the IP3023 processor can mingle instructions from eight different threads together in its pipeline *simultaneously*. Each stage can be working on an instruction from a different thread. Context switching on the IP3023 requires zero instructions and zero clock cycles. The threads can be synchronous hard real-time processes or non-real-time processes, in any combination. It's almost like having eight microprocessors on a single chip. The total amount of processing power available to these threads is 250 MIPS at 250 MHz. (Note that the IP3023 architecture supports up to 32 threads, so future implementations may deliver even greater performance.)

Intel is implementing a similar technique called "hyperthreading" in its Pentium® 4 processor. Hyperthreading allows a Pentium 4 to mix instructions from two threads in its pipelines. Context-switching isn't deterministic enough for hard real-time applications. That's good enough for a PC processor, but to meet the demands of wireless applications, a MASI architecture processor has to do better.

Eight-way multithreading and zero-cycle context switching are possible with the IP3023 because it has eight complete register banks for storing thread states. As each pipe stage begins processing an instruction, it simply switches to the bank that holds the corresponding state information. Other pipe stages can be accessing other banks (or the same bank) at the same time. Each bank has 16 GPRs (general-purpose registers) and 8 address registers, all 32 bits wide, plus a special "Source-3" register for some instructions that use three 32-bit operands. Each bank also has a 48-bit accumulator for storing results of multiply-accumulate instructions and a complete set of control registers, including an independent program counter.

To make high-performance software I/O a reality, MASI-based processors must schedule software I/O tasks with hard real-time determinism. Those threads always get higher priority. If a hard real-time thread needs 50 MIPS to get the job done – or one-fifth of the available 250 MIPS – the processor ensures that every fifth instruction is from that thread. Instructions from non-real-time threads get the remaining time slots, based on their priorities or a round-robin scheme. In a typical Internet gateway application, the processor might assign four threads to software I/O, leaving four threads available for other tasks. Nonreal-time threads can use any cycles available when a hard real-time thread is idle, so no performance is wastedReferring back to Figure 4, the architecture pipeline diagram, you'll notice that the first stage is devoted to thread scheduling. You won't find that stage in most other processors. Other processors blindly fetch the instruction at the next address indicated by their global program counter, no matter which thread the instruction belongs to. If the fetched instruction doesn't belong to a higher-priority thread, the processor must stall the pipeline and execute a time-consuming context switch.

Thanks to the thread-scheduling stage at the front of the pipeline, the IP3023 processor always fetches instructions based on which of the eight program counters (and therefore, which of the eight possible threads) currently enjoys the highest priority. It doesn't matter if the next instruction is from the current thread or a different thread.

Remember: Instructions from different threads can share the pipeline and there's no context-switching penalty. The IP3023 processor can switch threads on every cycle! Figure 5 illustrates this mechanism.

Not every thread, even a high-priority thread, needs to execute all the time. Wireless network traffic tends to be very bursty. Long idle periods are occasionally interrupted by sudden streams of packets. So the IP3023 processor can suspend a thread indefinitely until an interrupt brings it back to life.

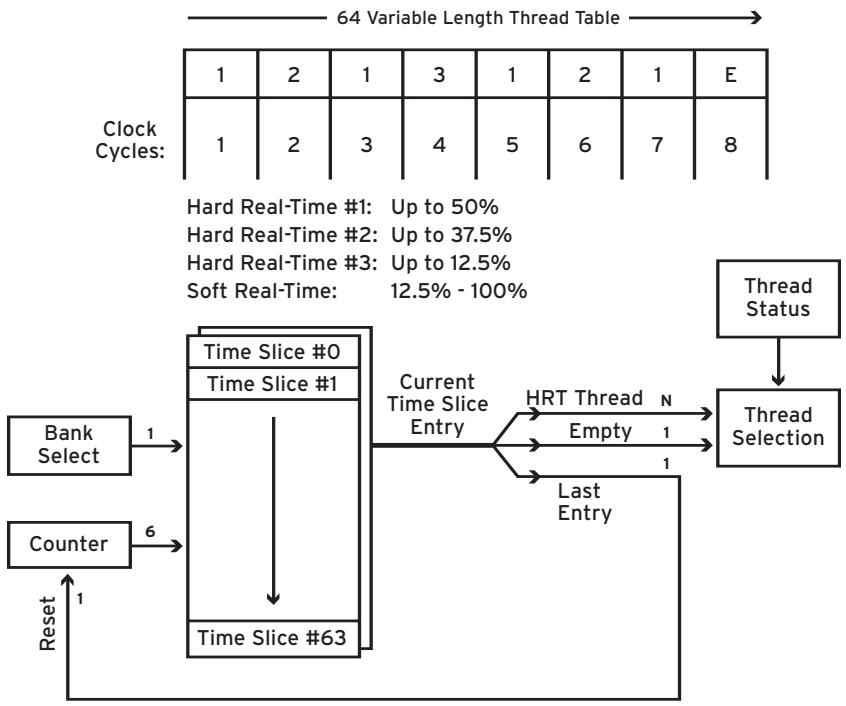


Figure 5: The IP3023 architecture manages instruction-level multithreading in hardware.

Interrupt response is incredibly quick: switching threads requires only one clock cycle, or 4 ns (nanoseconds) at 250 MHz. A conventional processor and multithreaded RTOS running at the same clock rate would need about 100 clock cycles (400 ns) to switch contexts and might allocate 1,000 clock cycles to each task. Non-real-time operating systems such as Linux® and VxWorks® are even slower. Conventional systems have to spend more time on each task in order to amortize their heavy context-switching penalty over more clock cycles. Due to instruction-level hardware multithreading, there's no penalty and nothing to amortize.

Another big advantage is that the Ubicom OS never turns off the interrupts, as some other OSes (e.g., Linux) often do while handling a previous interrupt. The processor doesn't need a breather. It can run multiple interrupt handlers simultaneously in different hardware threads. When the 16-bit FIFO buffer in a serializer/deserializer (SerDes) receives 16 bits of packet data from the I/O pins, the SerDes triggers an interrupt. One cycle later, the IP3023 processor can be executing the interrupt handler. If the second SerDes triggers *another* interrupt for a different thread, the IP3023 processor can start executing *that* interrupt handler on the very next cycle.

The IP3023 processor doesn't have to wait for one interrupt handler to finish before starting another. It doesn't have to waste time dumping and filling registers to switch contexts. It doesn't have to stall the pipeline, flush caches, or reset a global program counter to switch threads. And it doesn't have to wait for a whole packet to arrive before processing the packet, which is why it gets by with such tiny FIFOs. A MASI processor handles packet data right off the pins, in real time, with no fuss.

V. AN OPTIMIZED INSTRUCTION SET

Earlier we alluded to the IP3023 architecture's instruction set, which Ubicom created especially for wireless networking applications. This is a critical piece of the solution, because networking poses thorny problems for older CPU architectures that were designed many years ago for completely different applications.

RISC architectures were designed to speedily handle a few carefully defined data types carefully aligned on regular memory boundaries. Under ideal conditions, it's as precise as a highly trained army drill team on parade. Unfortunately, networking isn't an ideal application. Data packets are like a rabble of raw recruits who can't tell left from right. Depending on the protocol, packets can range in size from a few bytes to thousands of bytes. If they're aligned on an even memory boundary, it's probably an accident. The processor has to verify each byte for accuracy in case of transmission errors. Each packet has a header that the processor must strip off, scan, analyze, and act upon. These operations often require the processor to peel apart the bytes and shift bits around in odd ways.

It's no surprise that engineers at other companies have been scrambling to expand their already-bloated instruction sets with additional instructions for networking. Ubicom discarded all that baggage and designed a compact instruction set that focuses exclusively on packet processing.

We've already described how the IP3023 architecture instructions can execute direct memory-to-memory operations without shuffling data in and out of registers. The instructions also have other characteristics that boost performance, save clock cycles, and reduce code size:

- A rich variety of bit manipulations, including logical and arithmetic shifts, double data-type shifts (64 bits to 32 bits), bit-field extractions, bit-wise merges with masking, shift and merge to concatenate bytes, bit-set/bit-clear/bit-test on any register or memory value, and bit-reverse operations.
- Numerous addressing modes, including modes that use 8-, 16-, and 24-bit immediate values; register-based addressing; and memory-indirect addressing from a base address, a base + offset address (some using immediates), or a base + immediate-offset with pre- or post auto-incrementing.
- Powerful arithmetic instructions, including 16-bit multiply-accumulates (signed, unsigned, or fractional) that target a 48-bit accumulator, and a CRCGEN instruction for byte verification.
- 16- and 32-bit condition codes for branches. One type of conditional branch can jump to a 23-bit offset from the program counter. Subroutine calls can make 26-bit PC-relative jumps or memory-indirect jumps.
- Branch prediction is static to preserve real-time determinism, but the compiler can insert a one-bit hint to suggest that a branch will jump forward or backward. (Note that the IP3023 architecture doesn't need complex branch prediction to hide cache-miss penalties – there are no instruction or data caches.)
- No branch-delay slot is necessary, because instruction-level multithreading allows the IP3023 architecture to execute an instruction from a different thread immediately after a branch in the current thread.

This small, optimized instruction set is amazingly efficient for packet processing. With one or a few instructions, it can perform tasks that would require several instructions on other processors. For instance, a single shift-double instruction can align data to any bit position in memory and extract any 32 bits from the data. This is invaluable for manipulating data packets, which usually aren't aligned on 32-bit boundaries. Normally, the task would require several instructions to load the data from memory, mask it, shift it, and then store it back to memory. Likewise, the CRCGEN instruction applies any CRC method on one byte of data at a time, an operation that would normally require four to eight instructions and a 256-byte lookup table.

The IP3023 processor's memory-to-memory architecture is particularly useful for memory moves, whether the data is aligned or not. Here's an example of a 32-bit aligned memory move:

IP3023:
```
move.4 (a6)4++, (a5)4++
```

MIPS:
```
lw t1,0(t5)
sw t1,0(t6)
addu t5, t5, 4
addu t6, t6, 4
```

Although the IP3023 processor needs only one instruction to duplicate the work of four MIPS instructions, that example actually plays to the strengths of the MIPS processor, which is designed to handle 32-bit aligned data. If the data is unaligned – a common occurrence in packet processing – the IP3023 processor's advantage in bit-wise operations is even more apparent:

IP3023:
```
move.4 source3,(a5)
shftd d2, 4(a5)++, #shiftcount
move.4 (a6)4++, d2
```

MIPS:
```
ldw t1, (t5)
ldw t2, 4(t5)
sll t3, t1, shift
srl t4, t2, (32-shift)
or t7, t3, t4
stw t7, 0(t6)
addu t5, t5, 4
addu t6, t6, 4
```

Note that the IP3023 processor's SHFTD instruction is using the special Source-3 register in this case to specify the amount of the shift, effectively making SHFTD a three-operand instruction.

The IP3023 processor's instruction set also supports hardware multithreading by allowing precise synchronization among the simultaneously executing threads. For instance, there could be a problem if an instruction in one thread modified a memory location currently being manipulated by another thread. The second thread would no longer be working with valid data. And when the second thread stored its result back to memory, it would overwrite the value stored by the first thread. This is a common dilemma when designing multithreaded or multiprocessor architectures.

One solution is to lock a memory address currently being used by a thread so it can't be manipulated by another thread. This is often called a *spin lock.* Programmers can use the IP3023 processor's bit-set and bit-clear instructions as spin-lock operations because they set a condition code with their prior value. Other threads can check the condition code to see if the value has been modified.

In many cases, however, a spin lock isn't necessary. All of the IP3023 processor's memory-to-memory instructions are *atomic* with respect to other threads. They can read a value from memory, modify the value, and write the result back to memory as a single atomic operation that won't be interrupted by another thread. A practical example is a memory counter used by more than one thread. Each thread can increment the counter without worrying about interference from another thread – and without protecting the counter with a spin lock.

Although the IP3023 processor is not a Java processor, its instruction set and memory-to-memory architecture are well-suited for Java interpreters. The Java virtual machine is stack-based; consequently, most other processors copy the top of the Java stack into their registers for faster access. But unlike register files, stacks are unbounded in size, so the Java stack belongs in memory. The IP3023 processor readily accommodates a memory-based stack of variable size because its memory-to-memory architecture allows a single instruction to pull a value off the stack, perform an operation, and push the result back onto the stack. This dovetails with Java's bytecode instructions, which also operate in a memory-to-memory fashion. Ubicom has successfully tested the core of a Java virtual machine on the IP3023 processor.

VI. THE SOFTWARE I/O ADVANTAGE

By far the most important difference between Ubicom's approach and other wireless network processors is the high-performance execution of software I/O. Software I/O allows the processor to flexibly support many different communication protocols simply by modifying the low-level software and perhaps a small amount of external hardware. Software I/O is better because a single chip can work with virtually any protocol likely to be encountered in a wireless application, today or tomorrow. It's even better for Ubicom's customers, because they don't have to develop custom ASICs or settle for an off-the-shelf chip that locks them into a particular protocol – a critical consideration in an era when industry standards are rapidly evolving. And if a protocol changes, customers can update the software implementation in the field over the Internet, instead of recalling the product and redesigning the chip.

Avoiding an ASIC project can cut 12-24 months off the design cycle and save hundreds of thousands of dollars in NRE (nonrecurring engineering) costs, expensive mask sets, tedious re-spins of silicon, and costly chip manufacturing. Developers don't have to shop around for IP (intellectual property) or struggle to integrate IP from different providers. The time-to-market advantage is obvious. Using software I/O technology, it's even possible to design and deploy products whose firmware can be upgraded remotely over the Internet to support new or revised protocols on demand.

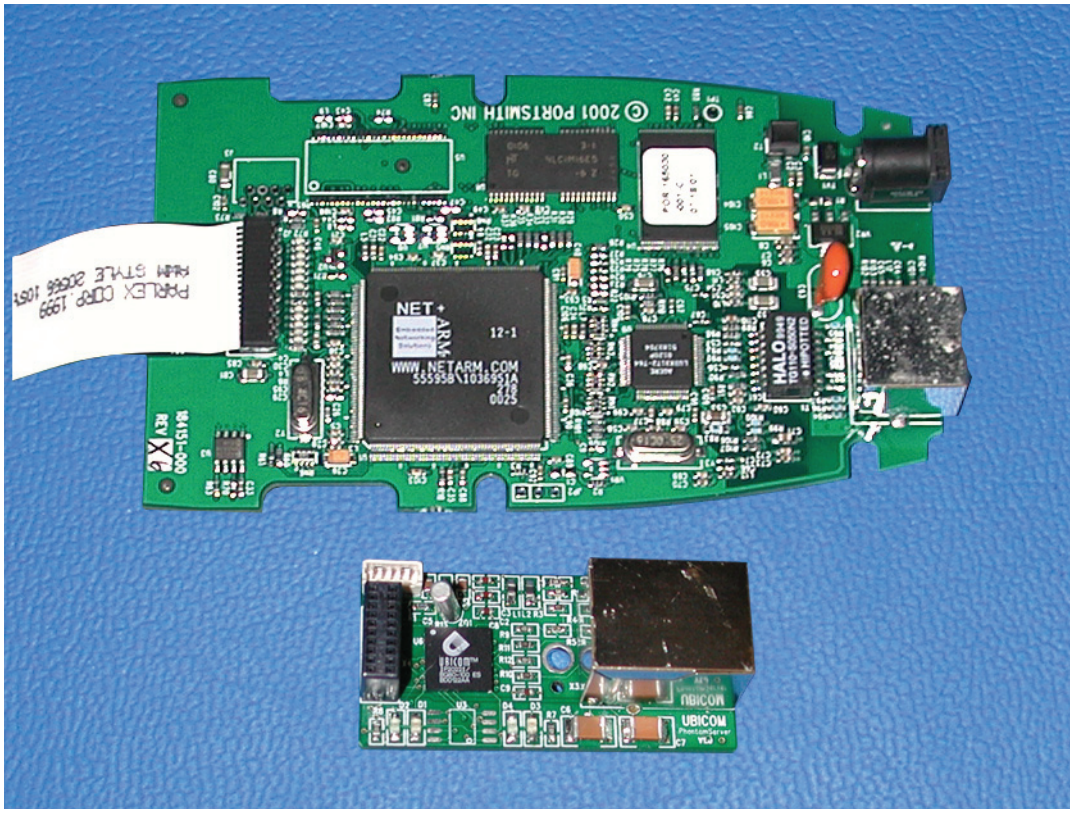


Figure 6: A Ubicom-based solution can be much smaller than competing solutions.

Software I/O technology can also reduce the system's chip count. Ubicom wireless network processors can implement the most common media access controllers (MACs), physical layer interfaces (PHYs), codecs, and baseband controllers in software, eliminating the need for redundant external chips. System boards can be smaller, simpler to design, less costly, more reliable, and more power-efficient.

The secret of software I/O technology is a processor that's powerful enough to decode virtually any wireless or end-node wired protocol in software, and perform other necessary functions, while handling the incoming packets at wire speed. Brute force could get the job done, but a large general-purpose processor running at high clock frequencies would be too expensive and power-hungry. A better solution is an integrated chip designed from the ground up as a wireless network processor. MASI-based processors can perform those functions in less silicon by implementing I/O in software.

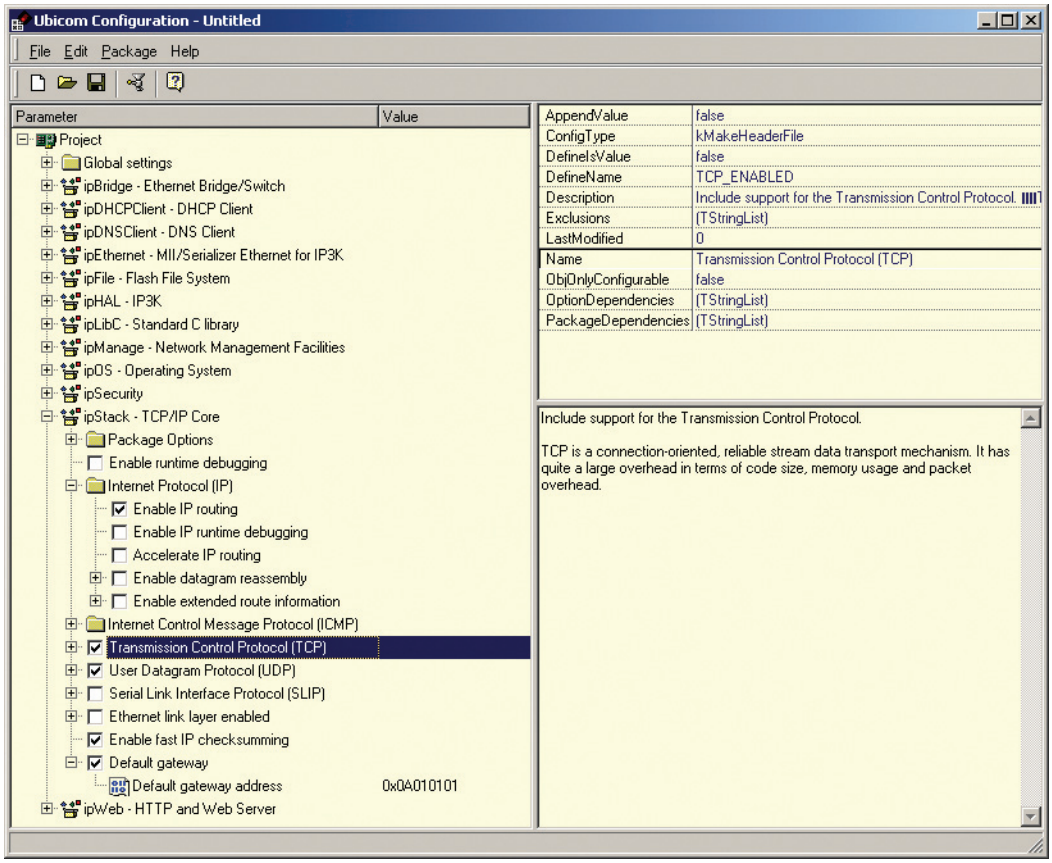


Figure 7: The Ubicom Configuration tool lets developers rapidly choose and configure a set of communication protocols using software I/O technology.

Figure 7 shows how developers can use the Ubicom Configuration tool to instantly configure IP3023 processors with the desired communication protocols. Prebuilt low-level protocols include TCP/IP, UDP, DHCP, NAT, and SNMP. (Protocols not yet supported by the Ubicom Configuration tool are easily adapted from existing code.) The Ubicom Configuration tool automatically generates the header files, make files, and #defines for the compiler. The compiler generates an executable file optimized for small code size rather than speed, because IP3023 wireless network processors have plenty of performance for wireless applications. The executable image of the whole stack – including the Ubicom operating system – is ready to download into the system's flash memory.

The ease of this process allows developers to focus on higher-level features that will differentiate their products from the competition. Developers don't have to waste time reinventing the wheel by creating protocols, network stacks, MACs, and RTOSes from scratch. There's little room for differentiation at the stack level, anyway, and it would only add months to the design cycle. It's much easier to assemble the stack using pre-verified, optimized building blocks already mated to the processor architecture.

To implement I/O in software, IP3023 processors have unusually versatile I/O capabilities and support logic. There are 106 GPIO (general-purpose I/O) pins that developers can use for various purposes. Here are some possibilities:

- Two serial I/O ports (8 GPIOs each) for USB 1.1 master/slave interfaces, GPSI (general purpose serial interface), SPI™ (serial peripheral interface), UART, or a BlueRF™-compatible interface to an external Bluetooth radio chip. In addition, one of these ports can be used with the internal 10Base-T Ethernet PHY.
- Four MII (media independent interface) ports (16 GPIOs each) for a 10/100-Mbps Ethernet PHY, HomePlug PHY, or external host processor. These ports can also configured as PHY-side MIIs, so IP3023 processors can function as a peripheral chip.
- A Utopia interface for an ADSL modem (24 GPIOs).
- A 16-bit SDRAM interface or PCMCIA interface (8 GPIOs).
- An 8-bit flash memory interface (32 pins).
- An 8 MHz CardBus, 802.11a, or 802.11g interface (48 GPIOs).
- A PCM (pulse code modulation) interface for telephones (8 GPIOs).
- SPI debug port.
- Developers may configure any unused ports as GPIOs.

To manage the serial traffic, IP3023 processors have two SerDes blocks, enabling two full-duplex channels of serial I/O. Each SerDes has a 16-bit FIFO buffer. Figure 8 is a block diagram of the internal resources of an IP3023 processor.

You'll notice from the block diagram that IP3023 processors also have some important support logic. IP3023 wireless network processors have a true random number generator for encryption algorithms; separate PLLs for the CPU core and peripheral I/O blocks; and various timers, including a watchdog timer. For power management, they have multiple power-down states, the ability to disable individual threads with a suspend instruction, and a software-controlled clock. It's doubtful that you'd find all those features on a general-purpose processor.

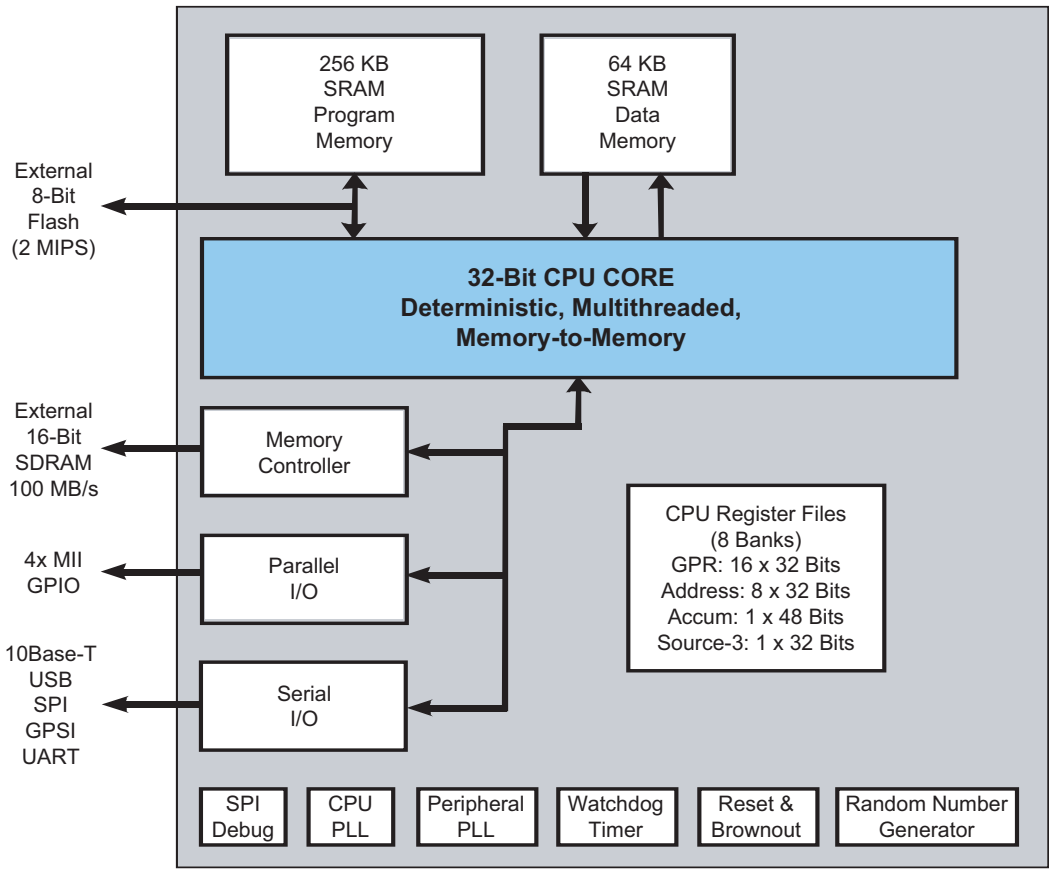


Figure 8: IP3023 wireless network processor block diagram

Consider how an IP3023 processor's I/O capabilities and software I/O technology can reduce the chip count in some typical wireless applications. In a wireless Bluetooth device, IP3023 processors can perform the baseband controller and MAC functions, interfacing directly to the radio chip. In a device connecting Bluetooth to an Ethernet network, IP3023 processors perform the Ethernet MAC and PHY functions in software. In a HomePlug device, IP3023 processors execute the HomePlug MAC and Ethernet MAC functions in software, interfacing directly to the HomePlug PHY. In a DSL gateway, IP3023 processors can use Utopia to talk directly with an ADSL modem chipset. In any device that requires a CardBus, PCI, or Mini PCI interface, IP3023 processors can execute those functions in software.

Don't overlook another major benefit of software I/O technology: Even after a product is deployed in the field, it's possible to upgrade the firmware to fix bugs, improve performance, adapt to evolving protocols, and deliver new features. The potential savings in tech support costs, product recalls, and on-site maintenance alone can justify a software I/O solution.

VII. CONCLUSIONS

The IP3023 wireless network processors can handle many communication tasks that would overwhelm a less-integrated processor or demand a much faster general-purpose processor. They perfectly embody Ubicom's philosophy – slash the chip count, board size, design complexity, power consumption, manufacturing cost, and design cycles of wireless products by offering highly integrated chips that are optimized for communications. To accomplish those goals, IP3023 wireless network processors have several key characteristics:

An efficient 32-bit MASI core designed from the ground up for wireless communications.

A memory-to-memory instruction-set architecture that reduces the need for on-chip packet buffers and external flash.

Software I/O technology for protocol flexibility, compact code, and low cost.

Instruction-level multithreading for high performance, low-latency packet processing, and deterministic real-time response.

Low power consumption via smaller silicon and versatile power management.

Outstanding designs like the MASI-based IP3023 processor don't happen by accident. They are the result of years of research and development and enable a combination of features unmatched in the marketplace.

635 Clyde Avenue
Mountain View, CA 94043
tel 650.210.1500
fax 650.210.8896
www.ubicom.com

Copyright © 2003 Ubicom, Inc. All rights reserved.
Ubicom, IP2000, IP3023, and ipOS are trademarks of Ubicom, Inc.
All other trademarks are the property of their respective holders.

WP-IP3023WNP-01 (6/11/2003)