Reed Electronics Group

MICROPROCESSOR REPORT

www.MPRonline.com

THE INSIDER'S GUIDE TO MICROPROCESSOR HARDWARE

XMOS Redefines Silicon

Software-Defined Chips Attack ASICs, ASSPs, FPGAs

By Tom R. Halfhill {8/6/07-01}

There seems to be no end to startups seeking alternatives to traditional processing technologies. Of course, the shortcomings of conventional devices are well known. ASICs are expensive, time-consuming, and risky to develop. ASSPs are available to everyone, offering less opportunity for differentiation. FPGAs are expensive to buy in large volumes and more difficult to program.

XMOS Semiconductor—a two-year-old fabless semiconductor company in Bristol, England—is pushing an alternative it calls "software-defined silicon." In this concept, a multicore array of general-purpose embedded-processor cores uses hardware multithreading to run the control software and application software under hard real-time constraints. At the same time, separate threads drive the chip's pins to emulate the required I/O interfaces—Ethernet, USB, UARTs, I2C, and so forth. This combination of multicore integration, deterministic multithreading, and software-defined I/O allows a general-purpose microprocessor to perform the functions of an SoC, but without custom acceleration hardware or dedicated I/O controllers.

The basic concept isn't new. A few years ago, Ubicom, a small Silicon Valley company, introduced chips that use deterministic hardware multithreading for packet processing and software-defined I/O. Ubicom's Multithreaded Architecture for Software I/O (MASI) brings more flexibility to standard-product silicon. (See *MPR 4/21/03-01*, "Ubicom's New NPU Stays Small.") XMOS hopes to carry the concept to higher levels of integration and performance (and sales). Some, if not most, XMOS chips will be multicore designs capable of implementing 100Mb/s Ethernet and even faster I/O interfaces in software while running a real-time operating system and performing digital-signal processing.

At this time, XMOS is revealing little about its underlying XCore microprocessor architecture. In July, the company emerged from stealth mode and began describing its technology in general terms. XMOS plans to divulge more details later this year, when the first silicon arrives from the foundry. The initial chip is a multicore design fabricated in a standard 90nm CMOS process. XMOS plans to begin selling chips next year at prices ranging from $1 to $10. At those prices, XCore processors would compete favorably with ASSPs and FPGAs—and, as standard products, they would eliminate the need for some customers to develop ASICs.

No RTL or Assembly Required

At the heart of XMOS technology is a proprietary 32-bit CPU architecture specifically designed for hardware multithreading, software-driven I/O, and high-level programmability. Although some XMOS engineers hail from ARM and other processor companies, XMOS says its CPU architecture doesn't imitate any existing architecture. It's a clean-sheet design, optimized for its duties, and it's not intended to be programmed in assembly language or register-transfer-level (RTL) languages. Even the device drivers and other low-level software that XMOS will provide are written in C, C++, and a derivative called XMOS C (XC).

XC adds timing statements, hardware I/O operations, hardware concurrency, and other extensions to the ANSI-standard C language. For instance, XC allows a programmer to specify that a certain operation will execute at a fixed

```
#include <assert.h>
#include "UartTransmit.h"

static unsigned g_bittime;
static unsigned g_metaNumWords;

void UartTransmitSetup(unsigned bittime,
unsigned metaNumWords)
{
       g_bittime = bittime;
       g_metaNumWords = metaNumWords;
}

unsigned UartTransmitByte(unsigned char byte,
unsigned time, outport txd)
{
       unsigned i;
       unsigned data = byte;

       i = 0;
       txd @ time =<< 0; // Start bit
       time += g_bittime;

       for (i = 0; i < 8; i += 1)
       {
              txd @ time =<<> data;
              time += g_bittime;
       }
       txd @ time =<< 1; // Stop bit
       time += g_bittime;
       return time;
}

void UartTransmit(unsigned char bytes[],
unsigned numBytes, outport txd, inport ctsTx,
outport rtsTx)
{
       timer t;
       unsigned time, i, index;
       int ctsTxVal;
       unsigned char byte;

       rtsTx =<< 0; // Assert RTS
       ctsTx =>> ctsTxVal; // Wait for CTS if needed
       if (ctsTxVal == 1)
       ctsTx =>> int ctsTx when (ctsTx == 0);
       t =>> time;
       time += g_bittime;

       for (i = 0, index = g_metaNumWords << 2; i
       < numBytes; i += 1, index += 1)
       {
              byte = bytes[index];
              time = UartTransmitByte(byte, time, txd);
       }

       rtsTx =<< 1; // Deassert RTS
```

Figure 1. This example XMOS C (XC) code implements a UART transmit function. Statements such as inport and outport refer directly to I/O operations on the chip's pins. References to time values can use internal or external timers as sources, and they can control I/O operations at specified times or intervals. Additional XC extensions support hardware-level concurrency, using either multithreading on a single processor core or multithreading and multitasking on multiple processor cores.

interval, such as once every 20ms. A new select statement is similar to a case statement, except that it vectors execution to a particular block of code whenever the chip receives specified events from an I/O interface. These and other extensions are necessary for hard real-time determinism and programmable I/O, which are central to the concept of software-defined silicon. Figure 1 shows an example of XC code.

XC doesn't go quite as far as other recent variants of C, such as System C. It's still a language primarily for writing software, not for describing hardware (except I/O interfaces). Programmers may use standard C or C++ to write software that doesn't require strict timing. Indeed, as Figure 2 illustrates, a typical project will have modules written in C, C++, and XC, depending on the purpose of the code. XMOS will provide new software-development tools capable of handling all project code within a single Eclipse-based integrated development environment.

In their source code, programmers explicitly assign threads to various hardware-I/O and software tasks. In effect, these assignments manually partition the XCore processing resources. The processors guarantee each thread a minimum level of performance, and threads consume no power when idle. With the help of cycle-accurate software simulators, programmers can determine at design time whether a particular I/O or software task is achieving the desired performance. If a task falls short of its performance target, programmers can assign it additional threads. If a particular XCore chip cannot deliver the required performance, even after allocating all its threads, developers can switch to a more powerful XCore chip that runs at a faster clock rate or has more processor cores.

XMOS is developing software libraries that will help developers implement popular I/O interfaces and carry out common software tasks. XMOS is also seeking to port existing software libraries to XCore. Preserving high-level programmability on an architecture that depends on strict timing is an important part of XMOS technology. It remains to be seen how effectively the XMOS development tools and libraries achieve this goal.

Hardware Multithreading Is Vital

To guarantee hard real-time performance, XCore processors use hardware multithreading, not conventional multithreading as implemented by an operating system. Essentially, hardware multithreading is deterministic time

Key People at XMOS Semiconductor

At today's semiconductor startups, expertise in software development and system software is as important as expertise in microprocessor architecture. Here are the backgrounds of three key people at XMOS Semiconductor.

James Foster: president, CEO, cofounder, board member. Foster was formerly CEO of Oxford Semiconductor and has held senior engineering and commercial roles at Lucent and Lattice Semiconductor. He has a BS in electronic engineering and an MBA from the Open University. Foster brings semiconductor industry experience and academic credentials to XMOS, as well as business knowledge.

David May: chief technology officer and cofounder. May formerly headed the Computer Science Department at Bristol University and worked for 16 years in the semiconductor industry, including such companies as Inmos and STMicroelectronics. May was the architect of the Inmos Transputer and helped design the Occam concurrent programming language. He has 33 patents for microprocessor technology, with more pending. In 1990, May was elected a Fellow of the Royal Society for his contributions to computer architecture and parallel computing. He is on the technical advisory boards of several semiconductor companies. As a pioneer in parallel processing and concurrent programming, May brings important hardware-design and software-development experience to XMOS.

Noel Hurley: vice president of marketing. Hurley was formerly director of CPU product marketing at ARM, where he was responsible for defining, promoting, and managing new products. Hurley also served in senior sales and marketing roles during 11 years at ARM. Previously, he was a senior applications engineer at Philips Semiconductors. He holds a bachelor's degree (with honors) in electrical and electronic engineering. Hurley's long tenure at ARM brings valuable knowledge about the embedded-processor market to XMOS.

slicing at the processor level. The processor can devote specific amounts of execution time to various tasks.

Each XCore processor core can simultaneously execute eight threads and switch thread contexts on every clock cycle. Although XMOS isn't yet releasing details about the XCore architecture, other processors implement hardware multithreading by maintaining duplicate register files for each thread. Instead of saving and restoring the contents of a register file during a context switch, the processor merely changes a pointer to the appropriate register file. There's no save-and-restore penalty for switching from one thread to another. XCore almost certainly uses this technique.

Threads execute in round-robin fashion, and the processor allots equal time to each thread. If eight threads are running, each thread gets 12.5% of the processor's execution time. If four threads are running, each thread gets 25% of the processor's execution time, and so forth. Note that for the purposes of this discussion, a "thread" isn't necessarily a subprocess within a multithreaded program—it could be a single-threaded process. For instance, one thread could be a device driver that drives the I/O pins to define a UART. Another thread could be a control program, such as a real-time operating system. Another thread could be a single-threaded application program. Still another thread could be a subprocess within a multithreaded application program.

Some processes may demand more performance than a single thread can deliver. In those cases, programmers may partition the task between two threads. For example, one thread might handle low-level I/O operations while the other handles the interface protocol and buffering. All threads continue executing in strict round-robin fashion. Some multithreaded processors, such as the new MIPS32 74K, have programmable thread-priority policies that accomplish hardware multithreading somewhat differently. (See *MPR 5/29/07-01*, "MIPS 74K Goes Superscalar," and *MPR 6/4/07-01*, "MIPS 74K Performance Update.") Both methods are valid; the XMOS method is simpler and more closely resembles Ubicom's model.

Simplicity is important, because XCore programmers must statically assign threads to specific tasks at design time, with help from the XMOS software-development tools. With an XCore processor, the basic unit of performance becomes a thread, not a clock cycle. Therefore, it's essential

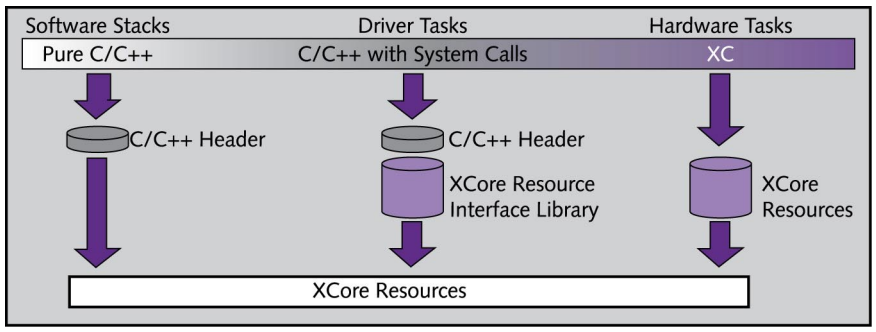

Figure 2. Developers working with an XMOS XCore processor can use standard C and C++ to write most of their software. Some software may call prewritten code libraries that XMOS provides. For modules that define custom I/O interfaces or that require strict timing, programmers use XMOS C (XC), which adds extensions to C. XMOS development tools unite all these C variants within an Eclipse framework.

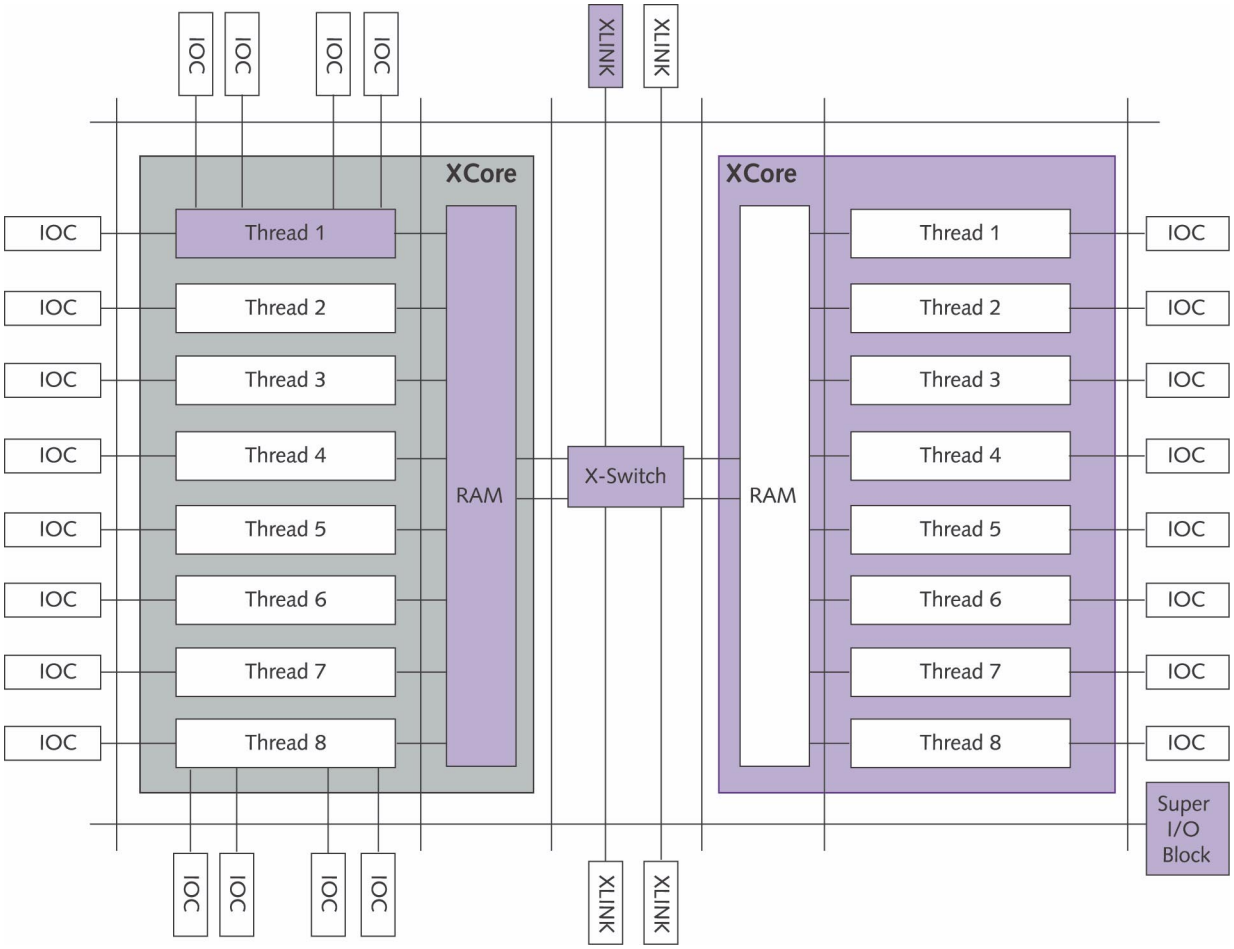

Figure 3. XLink is a new interconnect for XCore processors. It will unite multiple XCore processors on a single chip or span larger distances in multichip and multisystem designs. With an XLink network, hardware multithreading can span multiple cores or multiple chips.

for programmers and development tools to understand the performance available from each thread.

However, under software control, it's possible to dynamically reassign threads and rebalance workloads at run time. This capability opens up all sorts of possibilities. For instance, if an application doesn't need to use a particular I/O interface for a while, it can divert those threads to another task. An I2C interface could morph into a UART merely by swapping device drivers. Software-driven I/O is popularly called "bit bashing," but XMOS avoids the term, saying it evokes the bit-level manipulations performed by microcontrollers that have narrow I/O ports and registers. XMOS chips will have matching ports and registers that are 1, 4, 8, or 16 bits wide, reducing or eliminating the need for low-level bit masking.

Until XMOS tests its first silicon, the company is reluctant to release performance details. However, XMOS says initial XCore devices should execute 500 native mips per core—or about 62 mips for each of the eight threads. XMOS says this degree of performance will be sufficient to implement a 100Mb/s Ethernet interface or even a High-Speed USB 2.0 interface (480Mb/s) in software. For DSP applications, a 16-tap finite impulse response (FIR) filter will be capable of processing about seven million samples per second.

Multiple Processors Multiply Performance

Some applications will demand more performance than a single XCore processor can deliver, so another important piece of the technology is support for multicore chips and multichip systems. A proprietary XMOS interconnect called XLink allows multiple XCore processors to communicate with each other. XLink networks can coordinate multiple threads on a core, multiple threads on different cores, or multiple threads on different XCore chips.

XMOS plans to release more details about XLink this fall. As Figure 3 shows, XLink appears to be a crossbar bus that joins two or more processor cores together in a tight relationship. The compiled object code is position independent, so developers don't have to know which core a specific thread will run on—the software can distribute threads at run time. However, if programmers wish, they can assign specific threads to specific cores in advance.

On-chip interconnects have different requirements than chip-to-chip networks do. To accommodate the microscopic distances of on-chip interconnects and the board-scale distances of multichip systems, XLink runs a single logical protocol on different physical interfaces. XMOS development tools will need some knowledge of the network latencies to distribute the threads intelligently. The tools will probably rely on design-time software simulations to estimate the XLink latencies in multichip or multisystem networks.

Price & Availability

XMOS Semiconductor plans to introduce the first XCore chips next year at prices ranging from $1 to $10. Additional details will be announced this fall.

For more information visit: *www.xmos.com*.

Chips Aren't the Whole Story

Although XMOS is withholding details about its proprietary CPU architecture for now, those details aren't terribly important. If XMOS delivers on its promises, customers won't have to bother with those details, because XCore is designed for high-level programming at the thread level. Of course, it will be interesting to learn how XMOS took advantage of its rare opportunity to create a new CPU architecture optimized for hardware multithreading on multicore chips. But, given the fact that software-defined silicon isn't a new concept, the crucial question is whether XMOS can create a product line of useful, affordable chips adequately supported by processor cores, interconnects, and software-development tools.

One hurdle is XC. Developers tend to resist vendor-specific extensions to C, especially when those extensions bind their software to a proprietary CPU architecture launched by a startup. XMOS recognizes this hurdle and is writing system libraries of low-level device drivers and I/O interfaces in XC so that developers can write most of their application software in standard, portable C or C++. In addition, XMOS hopes to port some widely used software libraries to XCore. XMOS says it created the XC extensions only because there's no industry-standard open alternative—a common complaint.

Another hurdle for XMOS is software-development tools. XMOS is promising a lot for its tools. If thread allocation isn't stone-dumb simple, the chore will quickly get out of hand as the level of integration rises. XMOS is already talking about "arrays of cores"—implying that the initial multicore chips will lead to manycore or massively parallel XCore designs. For customers, the quality of the software-development tools soon becomes more important than the capabilities of the hardware.

Fortunately for XMOS, the basic concept of software-defined silicon is fundamentally sound. Indeed, the evolution of computing is largely a process of migrating more functions from hardware to software. As processors grow more powerful, I/O functions that once required dedicated hardware (on or off chip) can be comfortably handled in software. Dedicated hardware is still more efficient in high-volume applications, but many end-user products never attain those volumes. Fierce competition is one factor—before a product can amortize its expensive ASICs, numerous imitators often appear. Another factor is rapid market turnover—many consumer-electronics products have shelf lives measured in months, not years. Industry standards keep progressing, too. A programmable chip that can magically morph its I/O interfaces at design time or even at run time can postpone early obsolescence.

Therefore, the outlook for XMOS depends on how well the company executes all facets of its broad strategy. Designing a multithreaded processor is almost the easy part. XMOS will succeed or fail on the strengths of its multicore integration, software-development tools, extended C language, and system software.