CIRCUIT CELLAR®
THE MAGAZINE FOR COMPUTER APPLICATIONS

SILICON UPDATE

by Tom Cantrell

Turning the Core-ner

Parallax is poised to bring multicore to the masses with its Propeller microcontroller. It looks like a mild-mannered microcontroller, but wait until you see what's under the hood.

From: PACT 2006 Publicity
To: tom.cantrell@circuitcellar.com

Subject: PACT '06: Abstract Submission Deadline EXTENDED by 48 Hours

PACT'2006 CALL FOR PAPERS
http://www.cs.virginia.edu/~pact2006/

International Conference on Parallel Architectures and Compilation Techniques. Seattle, Washington, September 16–20, 2006

Due to an unforeseeable hardware failure, the site hosting the PACT 2006 submissions was down for most of the day on Monday, March 27…

You've all heard the old saying that those who forget the past are doomed to repeat it. Well, in the case of computer architecture, it can be said that those who remember the past are doomed to repeat it too.

As I pondered my pile of punch cards some 30 years ago, little did I realize that the computer I was (ab)using, a mighty IBM 360/91, was a veritable architectural roadmap of things to come. Pipelining and cache—natch. Virtual memory—check. Superscalar—got it. Branch prediction and speculative execution—ditto.

Don't get me wrong. The fact that the old-timey room full of big iron has been shrunk to a tiny chip is a wondrous thing. Not to mention the fact that today's bargain-basement PC has more MIPS, megahertz, and megabytes than anyone dreamt possible back in the mainframe days.

Having incorporated all the historic architectural features, the silicon CPU performance improvements in recent years have mainly been driven by implementation (e.g., faster clock and more cache). Indeed, there are arguments that bloated designs (e.g., Itanium) have overshot the sweet spot on the architectural price/performance curve. The evermore expensive and power-consuming architectural tweaks seem to return little in terms of real-world (i.e., existing application program) speedup.

As with the big-iron mainframes of yore, today's big-silicon mainframes-on-chip are running out of gas. What to do?

Where better to find the answer than at the first annual Multicore Expo? The premise is simple enough: Why not just use a bunch of simple and efficient processors rather than rely on ever-bloatier brainiacs?

The problem is that that's a simple premise that's been around a long, long time. I suspect there are few computer architects who, when pondering the single-CPU folly ahead, haven't hoped for a multicore miracle. Yogi Berra might say, "It's so simple that nobody's done it."

It's time for a change. On the computing front (e.g., PCs and video games), you're already seeing architects (Intel, AMD, Sony, IBM, and Sun) stick their toes in with multithread and multicore chips. The challenge here is getting the multicore chips to work well with traditional software applications and tools.

On the embedded front, there are more futuristic approaches that combine tens, hundreds, and perhaps even thousands of processors on a single chip. Here we find a lot of novel tool concepts as designers try to reconcile evolutionary migration with revolutionary aspirations.

There's a reason everyone stuck with the old single-core well past its

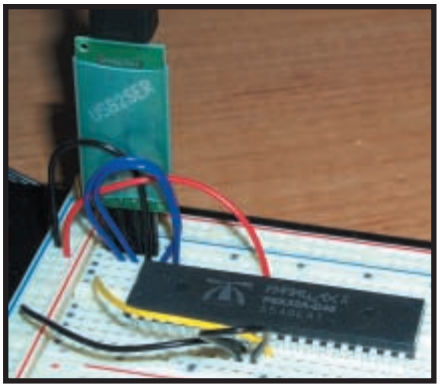

Photo 1—*Wiring up a minimal Propeller system configuration is trivially easy. Thanks to the on-chip oscillator, all that's needed is a connection for downloading code, either a USB-based debugger (shown here), a serial EEPROM, or both.*

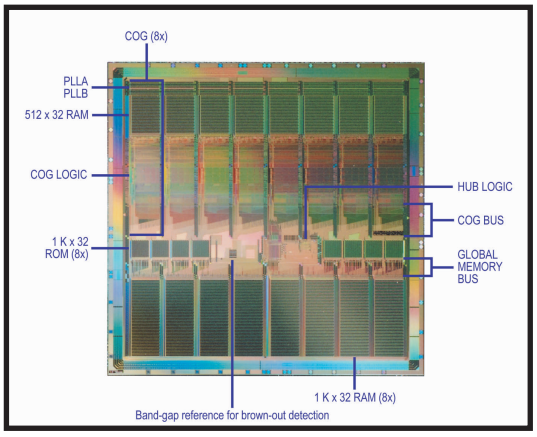

Photo 2—*Like the Cobra racecar of yore, Propeller crams a high-output eight-cylinder engine (i.e., eight cogs) in a small chassis to blow the doors off conventional MCUs.*

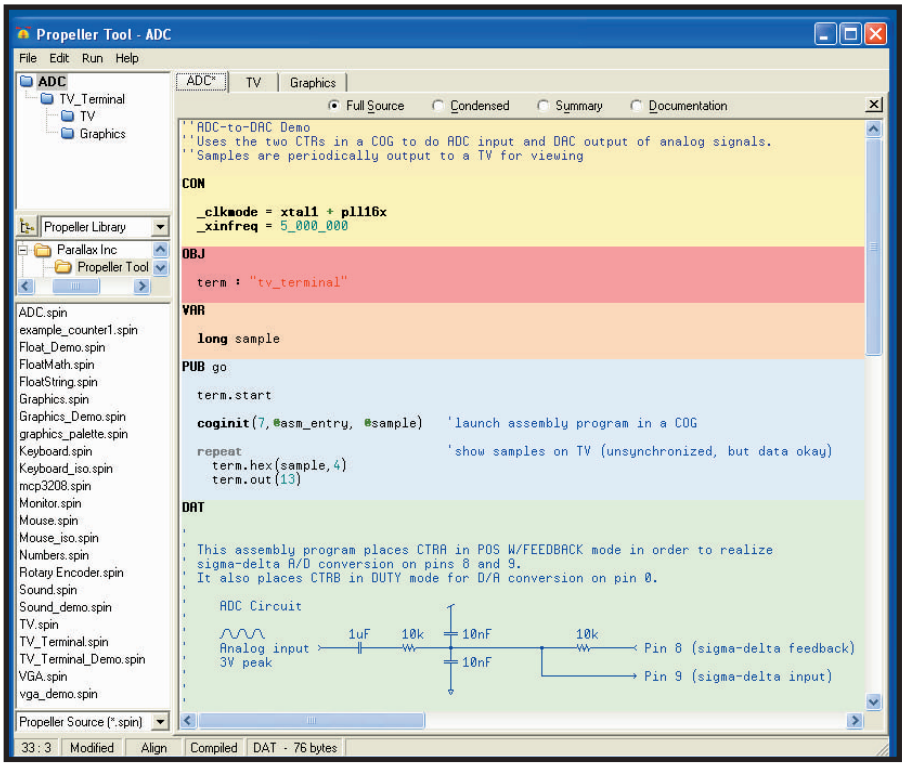

Photo 3—*Key to the Propeller philosophy is that the IDE encompasses all aspects of a design. Thanks to a special font that includes graphic symbols, program listings can even include hardware schematics and timing diagrams.*

prime. As baroque as the latest single-core CPU architectures are, multicore ups the complexity ante considerably. Traditionally, all you had to worry about was choosing from different architectures. Well, now you still have all the different architectures, but you also get to fret over numerous ways (e.g., bus, network, and pipeline) to connect them. Then there are the various intercore communication protocols to choose from (e.g., shared memory, message passing, and distributed objects). Finally, throw in all the new tools and languages that attempt to lay bare the pesky parallelism that's so hard to find.

Try to juggle all the options and you'll soon discover this multicore stuff can be pretty complicated. As I chatted with one of the Expo exhibitors at the reception, I joked, "beer and multicore don't go together," to which my burnt-out booth buddy replied, "Oh, beer helps—a lot." Look at the bright side: if it were easy, none of us would have jobs right?

Like most computer stuff, multicore isn't really a new idea. Just as that old 360/91 foreshadowed the brainiac microprocessors to come, the roots of multicore can be found in the boutique parallel-processing architectures (one example is described in James W. Moore's "The HEP Parallel Processor") that came to fruition during the Cold War (and many of which went down along with the Berlin Wall). These niche machines' big-iron footprints and price tags appear quaint in retrospect, but that shouldn't hide the fact that their designers had the courage to challenge the status quo of conventional computer design.

The difference between now and then? Moore's law means that multicore no longer requires a roomful of hardware and a number of significant digits have been slashed from the price tag. That's good news because another difference is that the single-core brainiac approach has run out of gas. Multicore is no longer an option; it's the only way out.

PROP JOB

This month, let's take a look at a new multicore chip from an unlikely source. Parallax is known as a supplier of low-cost and easy-to-use gadgets, most notably their line of BASIC Stamps. They're also the source for a high-performance, albeit conventional, 8-bit MCU, the 50-MIPS SX.

Now their creativity has found a new outlet in the form of the Propeller chip. It's an MCU that by virtue of a mere $10 price tag brings multicore to the masses, ready or not.

From 50,000′ there's little to distinguish the Propeller from conventional MCUs. Indeed, the 40-pin DIP package version has a positively retro feel (surface-mount 44-pin LQFP and QFN versions are also available).

Indeed, belying the black magic under the hood, the chip is embarrassingly easy to get up and running. As you can see in Photo 1, I got on the air with little more than a few connections to the USB2SER PC interface. Notice that a minimal configuration requires zero external components, not even a crystal thanks to an on-chip RC oscillator option. A more typical stand-alone setup will add a serial EEPROM to hold the application software and perhaps a crystal that works with an on-chip PLL to generate a faster precision clock (see Figure 1).

From the outside looking in, the most notable Propeller features are the ones that are missing. For example, where are all the UART, SPI, and counter/timer connections that crowd the typical MCU's pinout? For that

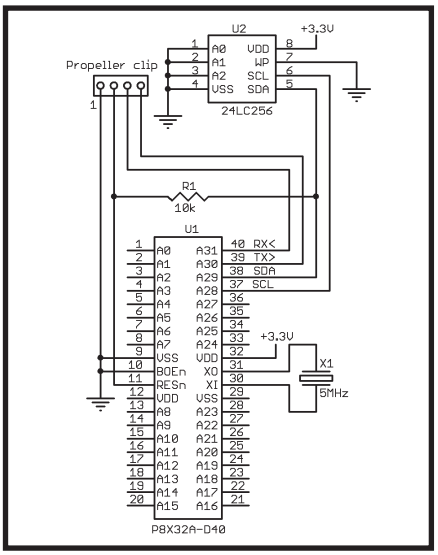

Figure 1—*The only truly dedicated Propeller pins are reset (RSTn), brownout enable (BOEn), and connections for an optional crystal (XI and XO). A typical system will also use two pins for the debugger interface (Rx and Tx) and two pins for an EEPROM (SDA and SCL) holding the application code.*

matter, where are the interrupts that are the hallmark of real-time embedded applications? Your eyes don't deceive. Other than the four pins used for the boot EEPROM and PC interface, Propeller I/O lines are all completely general-purpose.

Why doesn't the Propeller have dedicated I/O and interrupt pins? The short answer is that it doesn't need them. For the long answer, read on.

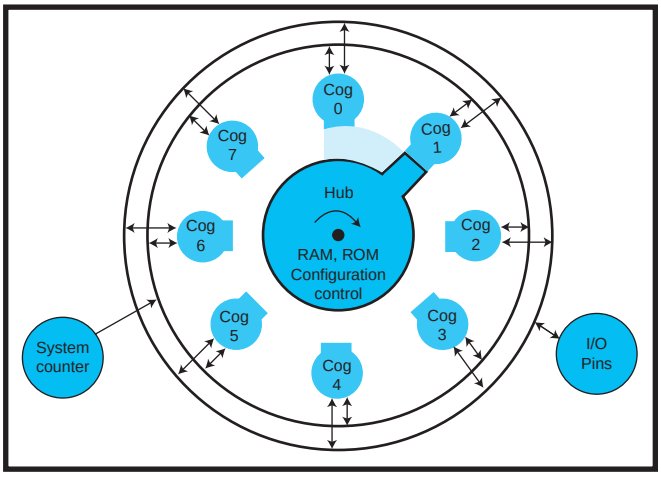

Figure 2—*Multicores are fine when each core is doing its own thing. The challenge arises when they contend for access to shared resources. Propeller uses a round-robin hub that grants each core deterministic access to shared RAM and ROM.*

COG IN THE MACHINE

For now let's keep it simple by getting to the core of the matter. Dig deep enough into the silicon and you'll find a so-called "cog" consisting of a tiny 32-bit processor with 2 KB (512 32-bit long words) of RAM and a dash of specialized video/timer hardware.

The cog itself is an interesting mix of RISC, CISC, and every other kind of 'ISC you can think of. Conventional machines differentiate between registers and memory and code and data. By contrast, with the cog, everything lives in the 2-KB RAM. And while there are only 64 opcodes in the instruction set, many the usual suspects, there are a number of unique embellishments. For instance, every instruction features conditional execution based on the state of the zero and carry flags. This supports deterministic straight-line coding instead of jittery conditional branches.

Similarly, but on the other side of the coin, writing the results (flags, destination register) of instruction execution is also an option. In a sense, the conditional execution and optional result writing mean the chip really has 8,192 instructions (i.e., 6-bit opcode + 4-bit conditional execution + 3-bit optional result writing), although many permutations would be of dubious use.

An interesting point about the stack is that there isn't one. Instead, the cog takes advantage of the fact that instructions are in RAM to emulate a conventional processor's stack-based CALLs and RETURNs by jamming the appropriate addresses into a JMP instruction at run-time (i.e., self-modifying code).

The video/timer stuff is pretty black magic, going well beyond the simple units found on typical MCUs. Suffice it to say that the two 32-bit counters with dedicated PLLs and video shifter can do very interesting things (e.g., audio and video) very quickly and very precisely. Each counter uses zero, one, or two pins depending on which of its 16 operating modes is selected.

HUBBA-HUBBA

At this point, you're probably thinking: "Looks pretty simple. What's the big deal?"

The big deal is found in Photo 2 (p. 80). The Propeller packs eight cogs worth of multicore machismo underneath its otherwise mild-mannered MCU exterior. Time to cue the *The Twilight Zone* music because now it starts to get a little spooky.

In techspeak, Propeller is a symmetric multiprocessor, or SMP (i.e., the cogs are all the same), using "shared memory" as the communication medium. Said shared memory, comprising 32 KB of RAM and 32 KB of ROM, is found in the hub.

The mechanism by which the memory is actually shared is invariably one of the messier aspects of multiprocessor design. A traditional approach has individual processors contending for access to the memory when they want it with an arbitration mechanism imposed to resolve conflicts.

Now, I suppose arbitration is better than a jury trial ("Ladies and gentleman, my client was unfairly denied access."), but it's still messy. First, the arbitration logic resides in the critical path between processor(s) and memory, thus slowing everything down. Second, it introduces timing uncertainty for all processors depending on the arbitration outcome (i.e., whether a processor is immediately granted access or it has to wait for another processor to finish). Finally, although not a requirement, arbitration schemes often lead down the primrose path of architectural embellishments such as priority (some processors have more access rights than others), which themselves lead to potential problems (e.g., priority inversion), calling for more hack workarounds (e.g., dynamically programmable priority). It's a death spiral of complexity, delay, and uncertainty.

By contrast, the Propeller sharing scheme is brutally simple. Like the distributor in an old V8, the hub simply goes round and round granting access to each of the cogs in turn (see Figure 2). The obvious downside is that cogs get access even if they don't need it, blocking others that possibly do. However, the distributor approach minimizes the jitter (i.e., lack of determinism) that plagues traditional arbitration schemes.

From a cog's perspective, the only uncertainty involves waiting for the first access to shared memory as the distributor spins around. After that first access is obtained, cogs can schedule their subsequent activity knowing precisely when subsequent accesses will be granted—no ifs, ands, or buts.

The hub also includes eight semaphores. However, these have nothing to do with the basic sharing mechanism. The distributor itself guarantees there can be no sharing conflicts for a single (byte, word, or long word) access. Indeed, that guarantee is exploited to implement the semaphores themselves.

Rather, the semaphores are a way for applications to adjudicate shared access to higher-level structures (arrays and I/O) if necessary.

The hub is also where the clocks for the entire chip (cogs and hub) are derived and distributed. As I mentioned earlier, one option is an on-chip RC oscillator that offers nominal 12 MHz and 20 kHz selections. The 20 kHz option is useful as a sleepy mode because cogs only consume about 3 µA at that clock rate. The other option is an external crystal or oscillator, which feeds a programmable PLL, boosting the rate by up to a factor of 16.

Now is a good time to talk megahertz and MIPs. The first chips run at up to 80 MHz (e.g., 5-MHz crystal with 16× PLL clock multiplier). Virtually all cog instructions execute in four clocks, except conditional branches, which require four (branch taken) or eight (not taken) clocks. That means the performance for the entire Propeller chip approaches 160 MIPs, or roughly 16 MIPs per buck. Not bad at all.

SPIN CONTROL

Topping off the Propeller package is the aptly named Spin language. Like the cog architecture itself, Spin combines aspects of languages that have gone before. You might think of it as an interpreted semi-object-oriented superassembler.

I say semi-object-oriented because it incorporates some of the most useful features of that approach (e.g., objects can be shared and have both publicly accessible and private routines and data) while eschewing the more esoteric aspects (polymorphism, inheritance, etc.).

Having programmed in dozens of languages over the years, I'm generally well past the stage of nitpicking syntactic trinkets. Does it really matter whether an assignment statement uses a = or := symbol? However, one aspect of Spin that's sure to cause some head scratching is the fact that white space matters. For instance, the last statement of a loop isn't necessarily delineated by a `NEXT` statement or a squiggly bracket, but rather by a change in the level of indentation. As with the vagaries of any language (English comes to mind), it's something you have to get used to. But at first this will trip you up, especially when cutting and pasting blocks of code.

These days, the integrated development environment (IDE) is as important as the language itself. The Propeller IDE incorporates a number of

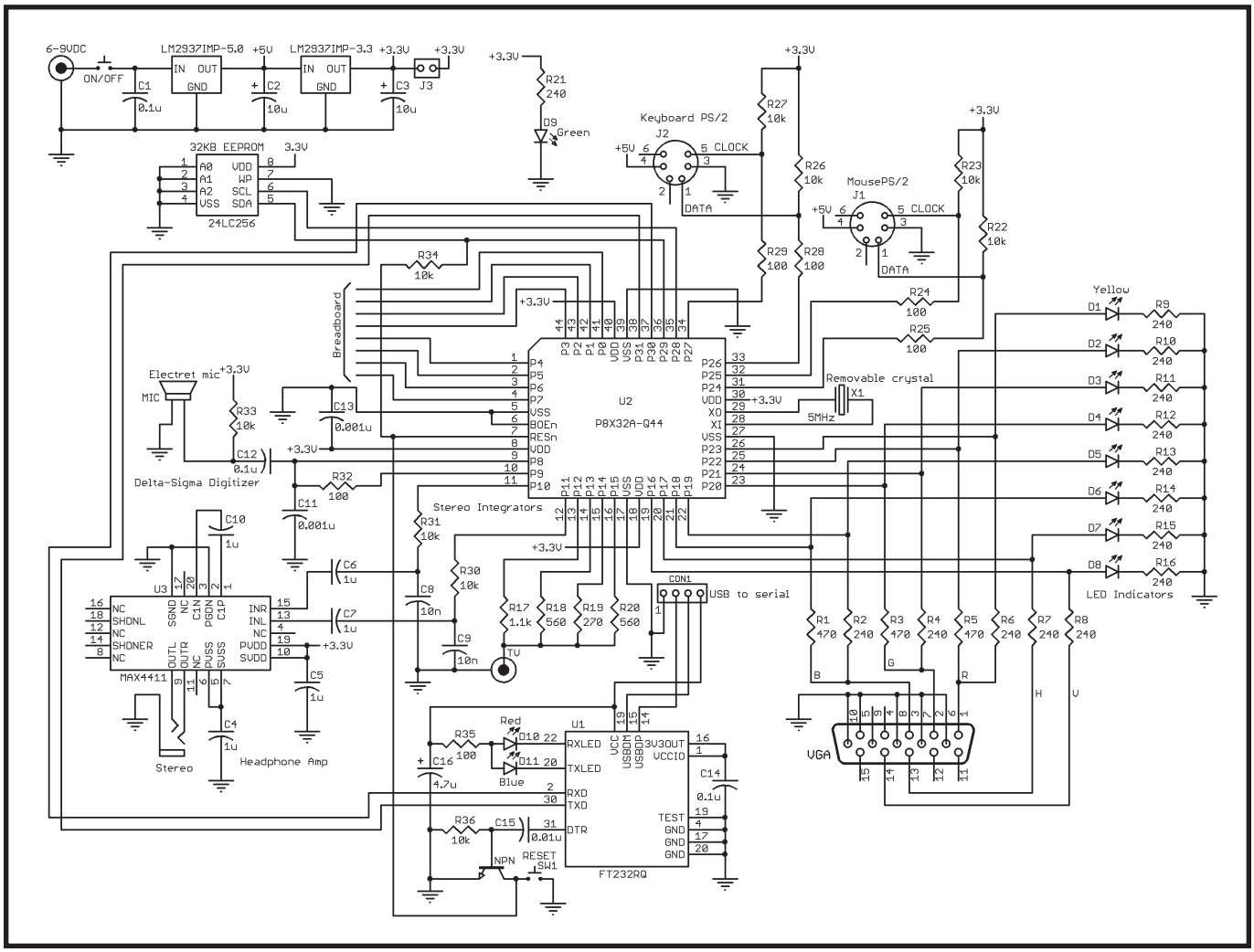

Figure 3—*The Propeller demonstration board demonstrates that you don't need a lot of silicon to get a lot of work (mouse, keyboard, audio, and video) done. The resistor-ladder DACs at the bottom support VGA and broadcast (!) video.*

novel features that reflect some interesting philosophical underpinnings. For example, there's a clear intention to support the collaborative development of objects (i.e., subprograms) for use by the entire community. To that end, programs are intended to be self-documenting, so much so that Parallax even defines a special font allowing the incorporation of schematics and equations as documentation in the code listing (see Photo 3, p. 81).

Similarly, different view modes in the IDE allow you to focus on the high level (i.e. documentation, public declarations) or zoom in on the gory details. There's even an archive option that packs all of a project's programs and the IDE itself in a ZIP file. That way, you can send it off to another person without having to worry whether they have the right version of the IDE, libraries, etc.

IT'S A COG'S LIFE

Following along as Propeller comes to life after reset helps clarify the big picture. The first thing that happens is that the chip looks for either an external EEPROM or PC to fill the entire contents of the on-chip shared RAM (32 KB) with all your application code (i.e., Spin and ASM programs).

Next, a copy of the tiny (2-KB) Spin interpreter is moved from the shared ROM (which also contains math look-up tables and a bitmap version of the documentation font mentioned earlier) into cog 0's 2 KB of local RAM. If you're inclined to complain that Spin doesn't have every language feature, consider the fact that it fits in less than 512 instructions and bite your tongue.

Next, Propeller starts executing the boot-up Spin program. What that actually means is that the interpreter, running in cog 0, fetches and interprets the first Spin program stored in the shared RAM. In other words, from the interpreter's point of view (running in cog 0), your Spin program is really data stored in the hub's shared RAM.

At this point, you're off to the races. That boot-up Spin program can spawn additional programs (Spin or ASM) on the remaining cogs. Spawned ASM programs are loaded and run in their entirety in the new cog. For spawned Spin programs, another copy of the interpreter is loaded into the new cog's RAM, where it (like the copy on cog 0) interprets Spin code in shared RAM. The fun part is that spawned programs themselves can spawn additional programs, including copies of themselves! With that in mind, one point to highlight is that Spin programs running on multiple cogs can use a single copy of common code held in shared RAM (i.e., multiple copies aren't required).

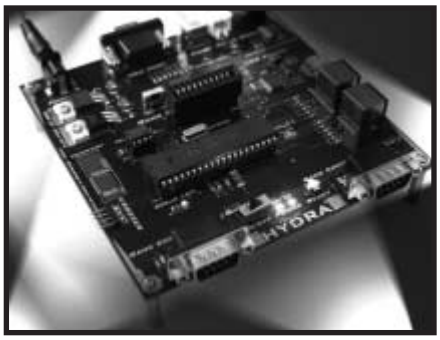

Photo 4—*Pac Man lives. Andre LaMothe takes advantage of the Propeller's unique capabilities to cram an entire video game platform into his Hydra board.*

PROPELLER HEADS WANTED

Already out of room and I've only scratched the surface. It's almost easier to define Propeller by what it isn't rather than by what it is.

What it isn't is the Holy Grail solution to the ages-old parallel problem (i.e., machines think in parallel, people don't). It has nothing to do with "automagically" parallelizing conventional software.

What it isn't is eight BASIC Stamps on a chip. Over time, enhanced tools and libraries will no doubt make Propeller more accessible. Nevertheless, it's nothing like a BASIC Stamp, nor is it intended to be.

As you might have gathered, it isn't a handholding solution for beginners. Propeller will not baby you. Tough love is more like it. There are a truly impressive number of ways to crash, and badly. Imagine different programs trying to use the same I/O pins or even messing around with the clock. You can do all of this and more—live free or die, as they say. Propeller comes with a huge length of rope, enough to do great things—and enough to tie yourself up in really tight knots.

So, what is Propeller? I guess what matters most, even more than the bits and bytes, is that it's the product of a unified vision. There's a purity of purpose from the loftiest heights of the IDE to the lithographic level of the transistors. And it's a vision not hamstrung by existing convention

As such, Propeller can deliver impressive results. I've been playing with the Propeller demonstration board, which, with little more than the chip itself, can handle mouse, keyboard, audio, and video, the latter including VGA, NTSC, and, with the right crystal, even broadcast (see Figure 3)!

For even more fun, check out the Hydra designed by Andre LaMothe (see Photo 4). It crams the equivalent of a circa '80s video game into Propeller, demonstrating just how far a little silicon can go in the hands of an expert. Also, supplementing the formal Propeller documentation, LaMothe's new Hydra book, *Game Programming for the Propeller Powered Hydra*, presents a lot of Propeller and Spin information in an accessible way.

One other thing Propeller for sure isn't: boring. When it comes to multicore, Propeller is proof that the fun has just begun.

Tom Cantrell (tom.cantrell@circuitcellar.com) has been working on chip, board, and systems design and marketing for several years.

PROJECT FILES

To download the hub block diagram, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2006/193.

RESOURCES

J. W. Moore, "The HEP Parallel Processor," *Los Alamos Science*, Fall 1983, http://www.fas.org/sgp/othergov/doe/lanl/pubs/00285915.pdf.

Multicore Association, www.multicore-association.org.

SOURCES

Hydra Propeller-powered SBC
Andre LaMothe
www.xgamestation.com

Propeller multicore microcontroller
Parallax, Inc.
www.parallax.com