

CIRCUIT CELLAR®

THE MAGAZINE FOR COMPUTER APPLICATIONS

FEATURE ARTICLE

Jim Turley

Design Your Own Microprocessor

Problem with your microprocessor? Complications with micros are often the case. But, what if you designed your own that did exactly what you needed it to? Jim provides a tutorial on creating a microprocessor. The bonus: You don't have to be Bill Gates to build it.



Microprocessors—you either love them or you hate them. If you hate yours, it's probably because it doesn't do what you want it to. Naturally, the bugs are the chip's fault. Don't you wish someone would make a processor that actually has the features you want?

Ta-da! Cue *Aladdin* music. Your wish has been granted. Now, you can design your own microprocessor without years of studying for your doctorate in Computer Science. You can be running code on your own microprocessor before the end of this week.

PROCESSOR FACTORY

Although building your own foundry is out of reach for most engineers (it costs more than \$1 billion just to upgrade an existing chip plant), it's easy to make a few of your own processors using low-cost FPGAs. Altera and Xilinx both encourage this behavior for the obvious reasons. In fact, both companies have entered the business of supplying soft processor core designs to encourage engineers to use more FPGA chips. Altera's Nios (see Figure 1)

and Xilinx's MicroBlaze (see Figure 2) are both free but only work, of course, in the parent company's chips.

Why put a processor in an FPGA? Why not buy a real microprocessor chip and solder it down? If you're going to use an FPGA for random logic anyway, you can eliminate the extra chip by putting the processor in the FPGA, as well. You also could experiment with different processors in your FPGA, evaluating various ones until you find the right balance of performance, size, cost, and power consumption.

The most interesting prospect, of course, is modifying the processor to suit your own whims. Or, dare I say, even designing your own processor from scratch (the first CPYou)? Well, you can, and there are some good reasons why you would (or would not) want to try this at home.

A ONE-MAN SHOW

For most engineers, designing your own processor architecture is a cool idea but not a practical one. A custom CPU has zero software support—no tools, no compiler, no debugger, and no one to turn to when bugs crop up. You're on your own.

Using an existing CPU, on the other hand, makes a lot of sense even if you're going to embed it into an FPGA or high-volume ASIC. In fact, according to Dataquest, the business of embedding CPUs into other chips is growing at about 25% per year and stood at about \$20 billion in 2000. That's no small potatoes. The average is 2.3 processors per intelligent ASIC (those with any programmability) and rising.

There's no shortage of processors to choose from, either. From little-known 8-bit processors created as university research projects to popular 16- and 32-bit processors like the '286, SPARC, ARM, or MIPS, the market is ready and willing to supply you with microprocessor intellectual property (IP). Some are free (and worth every

penny), while others might cost you as much as a new Mercedes.

One high-end free core is LEON-1, a SPARC clone distributed by the European Space Agency. This is a real SPARC processor, created with Sun's blessing, but without floating point capability or anywhere near the performance of Sun's current processors. Still, SPARC has good software tools (available for free) and a certain amount of appeal and prestige. LEON-1 is provided with VHDL source code (you supply the hardware compiler) and runs at 20 to 40 MHz in Xilinx or Altera FPGAs. At that speed, it would probably take you a week to boot Solaris.

At the low end (and also in the free category), are 8-bit processor designs from CMOSexod and Free-IP cores. Both designs have little 8-bit processors you've never heard of but come with free tools to get you started. The Free-IP Project also has a 6502 core, in case you've been hankering to resurrect your old Apple II. VAutomation also sells its 8-bit V8 processor with free tools. For your money, VAutomation gives you real technical support and the warm fuzzy feeling that their processor really works. They even use the V8 in some of their own products. The company also has a 6502 as well as Z80, 8086, and '186 cores. Now that Intel and AMD are both out of the

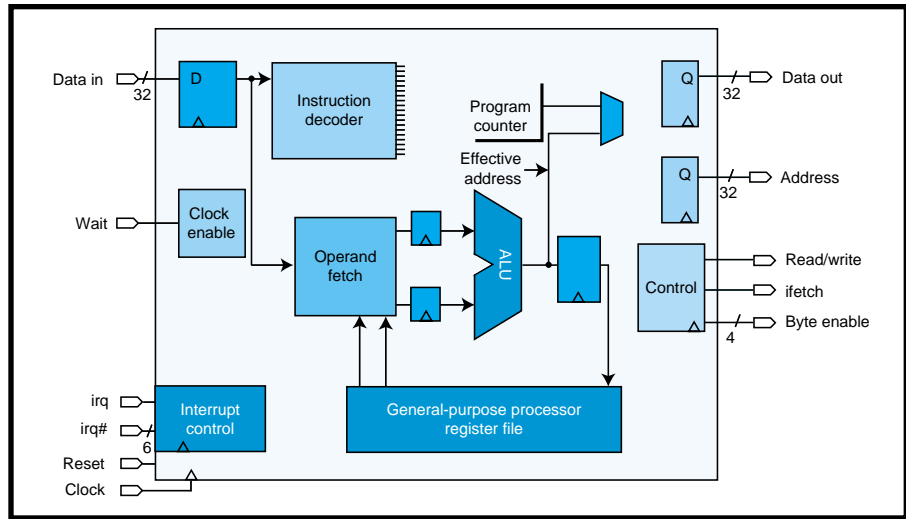


Figure 1—NIOS is a 16-bit processor Altera designed specifically to work in its own FPGA devices.

embedded 'x86 business, soft cores like VAutomation's may be your only way to get low-end 'x86 processors.

CLONE RANGERS

Any mention of the 'x86 architecture often brings up the issue of cloning. How can these knock-off processors exist? Are they legal? And, what liability am I exposing myself to if I use one? The answers are sometimes murky, but the short version is: don't worry about it. Cloning processors is a long and honored tradition, and the ones who got there illegally generally aren't around any more. Darwinian selection has culled the dubious ones.

Legal protection on processors is a funny thing. You can't patent an instruction set but you can copyright a processor's mnemonics as a work of literature. Even more bizarre is the fact that it's illegal to print a programmer's reference manual that's the same as, say, Pentium's, but it's perfectly okay to make a chip that runs Pentium code. That is, as long as you don't violate any hardware patents, which is where Lexra, picoTurbo, Shengyu Shen, and others may have run afoul of the technology lawyers.

Although you can't patent a processor's instructions per se, you can patent the circuits that make those instructions work. AMD, Transmeta, and other (now defunct) 'x86 clone makers have laboriously worked out different ways to get their processors to work exactly the same as Intel's, but without using Intel's exact circuitry. For a CPU as complex and baroque as Pentium, that's no small feat.

Cloning RISC processors is far easier, although not nearly as lucrative. RISC chips have, by definition, a reduced set of features so they're naturally less complicated. That simplicity was supposed to make streamlined RISC chips faster than tangled CISC chips like Pentium, but Intel's determination prevented that fairy tale from ever coming true. However, if you're going to clone a processor, RISC is definitely the way to go.

You can get a facsimile of a MIPS processor from OpenCores. Its "MIPS-lite" core supports most MIPS instruc-

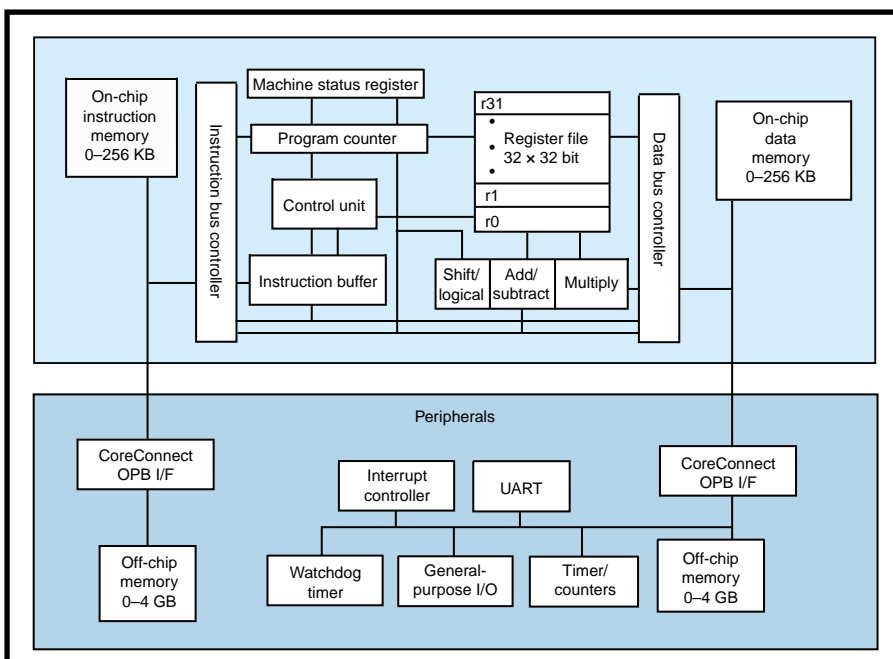


Figure 2—Xilinx's MicroBlaze, like NIOS, is an FPGA-only processor that resides in the "soft logic" of an FPGA.

tions from a few generations back (see Figure 3). For the premier MIPS knock-off, though, you needed to go to LEXRA Computing Engines. LEXRA's MIPS cores were close enough to the real thing to warrant two lawsuits from MIPS. Oddly enough though, neither suit alleged patent infringement, only copyright infringement. Until recently, LEXRA dodged both suits and sold its line of mostly MIPS-compatible cores. Now, LEXRA is officially MIPS licensed and concentrates on network processors.

Along those same lines, there were at least two unauthorized versions of the popular ARM architecture. A free one was available through OpenCores until last summer, when legal pressure apparently made it disappear. picoTurbo was also happy to sell you its version until it disappeared a few months ago.

YOU-BUILD-IT INSTRUCTION SETS

The next giant step just beyond building your own processor is designing your own processor. You can start from scratch and have all the glory (and all the headaches) for yourself, or you can use one of the many do-it-yourself (DIY) kits that are available. Like Lego blocks, these DIY microprocessors can be used to build an amazing array of completely different products.

The two big leaders in the area of user-configurable microprocessors are ARC International and Tensilica. ARC (not to be confused with ARM) is the older of the two and has more customers. Tensilica is based in Silicon Valley and seems to be catching on well. There are minor differences between ARC's ARCTangent and Tensilica's Xtensa processors, but they're more alike than different.

Both are 32-bit processors that you can tweak, twist, and modify to create your own private custom 32-bit CPU. You don't have to start from scratch either, because both processors work right out of the box. In this case, the box is a VHDL or Verilog description of the CPU. Neither company sells chips, only the recipe for the processor. Your freedom lies in your ability to adjust the recipe to suit your taste before you send your chip off to the



Photo 1—ARChitect is a program that allows engineers or programmers to design their own CPU with only a few mouse clicks.

silicon oven. You can make your chips in FPGA if you're looking at modest volumes, or go to a big Asian foundry if you're ready for millions of chips with your logo on them.

Both ARC and Tensilica start you off with a working 32-bit RISC core that's in the same league as ARM7, MIPS R3000, or most other mid-range 32-bit processors. These cores start out small enough to fit into all but the smallest parts from Xilinx or Altera. Both processors have the usual arithmetic (add, subtract, multiply), logical (shift, rotate, exclusive-OR), and flow-control (branch, jump, return) features that you'd expect to see. In short, they're completely functional RISC designs that crank out about 1.1 Dhrystone MIPS per megahertz, just like any self-respecting processor should these days.

But, that's just the basics. On top of the baseline features, you get to select from a menu of predefined, pre-tested options to extend your processor. Do you want a faster multiplier? Choose that option, and your chip will get smarter (and bigger). Do you need DSP features? Yeah, that's in there. Do you want bigger caches or more registers? Go for it. Both companies' cores have a dozen or so canned options you can choose from. Selecting them all might double the size of your basic processor core, but that's your choice.

Both firms use a familiar point-and-click user interface to configure the processor. ARC has a program called ARChitect and Tensilica's interface is available through its web site (see Photo 1). In either case, you click on the options you want, and after a few

minutes, your custom processor design is generated for you, complete with software tools to program it.

OH, MY VERY OWN INSTRUCTIONS

You don't see what you want on the menu? Now the real fun begins. Both companies let you go beyond the canned options and design your own features into the processor. You don't like the way the CPU handles overflows? Change it. Do you want to encrypt your code with a couple of funny instructions nobody can figure out? The limit is your creativity.

You can extend the ARCTangent using normal Verilog or VHDL to define hardware extensions to the processor. When ARChitect configures the ARCTangent core, it stitches in your custom instructions along with it. As long as you're familiar with either VHDL or Verilog, it's pretty straightforward. Tensilica does essentially the same thing but uses its own language, called Tensilica instruction extensions (TIE). With TIE, you define what you want the new features to do, then upload the definition to Tensilica's web site. Your wishes are integrated into the final design and downloaded back to you.

The end result is a processor that's part standard, part custom. Your CPU will be binary-compatible with other ARC or Tensilica processors, but with additions you define. What would you add to 30 years of computer architecture that hasn't already been done? Quite a lot, based on the evidence of customers who have done this.

Most users look for performance improvements, either by collapsing frequently used sequences of instructions or by designing custom instructions that do something unusual. If you're handling packed 8-bit pixels, a set of parallel SIMD add and multiply instructions would be helpful. If you're handling 24-bit audio samples, saturating arithmetic (math that clamps at certain levels) is a boon. Encryption guys love bit insert and extract instructions. Few of these operations appear in mainstream processors.

Other times, the goal is to compress code size. RISC, by its nature, has

lousy code density, but you can get some of that back by defining your own instructions. Other times, the objective has less to do with performance and more with competitive business. Custom instructions can prevent reverse engineering. Anyone can disassemble code for a standard microprocessor. All it takes is a few minutes with a logic analyzer or ROM burner. But if your processor has one or two custom instructions, disassembling code becomes much tougher. Even if your custom instruction is an NOP, no one knows that but you and sprinkling a few of them in your software will confuse the most diligent code spy.

However, if designing your own processor isn't advanced enough for you, try this out. New tools can design the processor for you based on a vague wish list. They're not perfect, and they're still at an early stage. But, we're gradually approaching the goal of a machine that executes the program "do what I'm thinking."

SOFTWARE IN, HARDWARE OUT

Celoxica has a hardware-design tool called Handel-C (see Figure 4). It's a language somewhat similar to Verilog and, you guessed it, C. It's more like C programming than Verilog modeling. The idea is to let programmers define hardware, even if they don't really know what they're doing. In fact, Celoxica considers it an advantage if you don't have hardware design experience. "Let Handel-C do the job, and you won't be disappointed," they say.

This concept is certainly appealing. Theoretically you can throw source code written in Handel-C at Celoxica's compiler and what will emerge is not a program but a hardware description of a circuit that executes that program. It's a kind of hard-coded processor tuned for exactly the program you gave it. Don't bother looking under the hood because you don't want to know what's there. Dyed-in-the-wool hardware designers might gag at the circuits that Celoxica's tools produce,

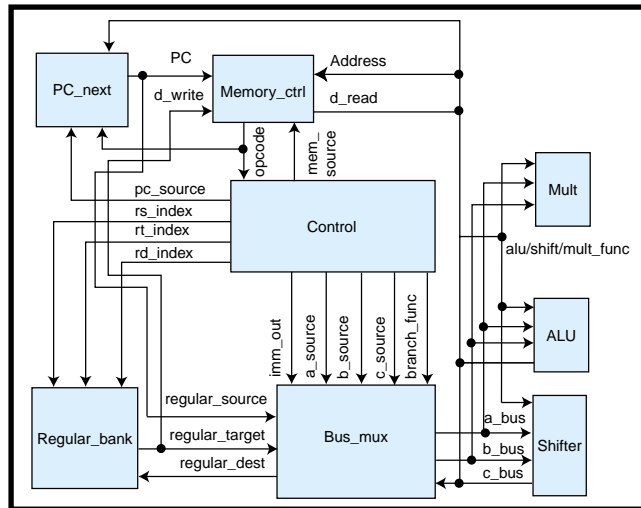


Figure 3—OpenCores' contributors have developed an open-source microprocessor that's similar to a MIPS core.

but that's not the point. A hardware engineer didn't produce it; an automated tool did. A company could theoretically dispense with its hardware engineers entirely and replace them all with C programmers armed with a few copies of Celoxica's Handel-C compiler. That's a scary thought.

In a similar vein, Agilent and STMicroelectronics have teamed up to produce a configurable processor with the mellifluous name of ST210.

(STMicroelectronics manufactures the chip, so they got to name it.) The ST210 is the first and, so far, the only example of a concept the group called PICO (program in, code out). The idea is to let automated tools make design decisions, rather than relying on engineers or programmers. As embedded systems get more complicated, the rea-

soning goes that hardware design is no longer intuitive. Humans don't always know what they want or what's best. The ST210 is a 250-MHz, eight-way, super-scalar VLIW beast, so it's a bit beyond the means of most embedded designers for the time being.

Along the same lines is Improv's Jazz processor. Like ST210, Jazz more or less defines itself based on the program it's asked to run. In Improv's case, you define the problem using a complex combination of Java programs and a visual-edit-

ing tool. From these, Jazz extracts what parallelism it can find and cobbles together the ideal processor for what you're trying to accomplish.

What all three of these approaches have in common is that the hardware is fixed. After the automated tool makes up its mind about what the processor should look like, that's it. From that point on, it's a fixed piece of hardware just like any other chip.

CHANGING RUN TIMES

Elixent takes the concept one step further. Yes, it has an automated tool that defines the circuitry from a source program, but instead of creating a processor, it creates multiple hardware implementations and switches among them on the fly based on the workload (see Figure 5). Elixent com-

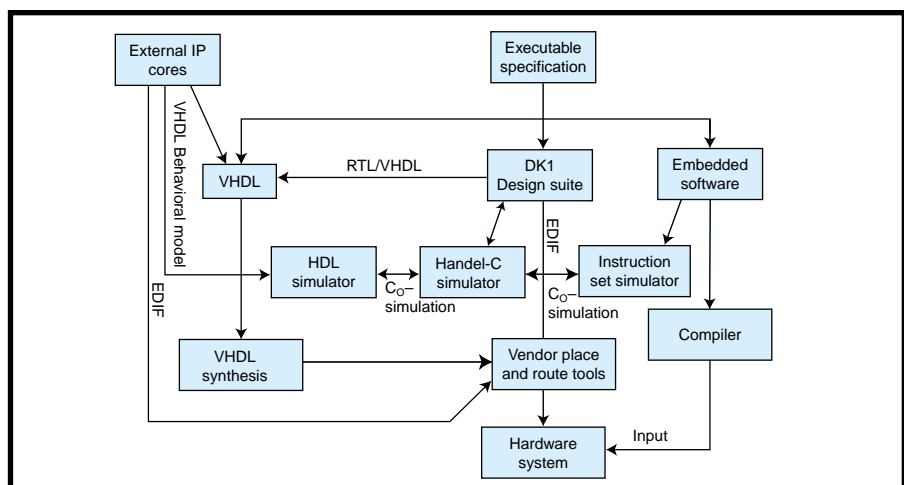


Figure 4—Celoxica's Handel-C is similar to C, but instead of producing a program, it creates a circuit design.

binesthe “software in, hardware out” concept of Celoxica with the real-time reconfigurability of an FPGA. The result is a programmable chip that can transmogrify itself for various tasks.

Elixent enters the world of dynamic reconfigurability, or reconfigurable computing (RC). RC seems like a completely reasonable concept: change the hardware on the fly to suit the application. After all, nearly all embedded systems handle different tasks at different times. One moment you’re decompressing audio streams, the next you’re handling an interrupt, and the next you’re accepting user input. It stands to reason that whenever one part of the circuit is working the other parts must be idle. This is often true, but there’s nothing you can do about it. Hardware systems have always been created from a superset of all the functions required over the life of the product, rolled into one.

If RC takes off, that might change. For example, future generations of cell phones may use half the transistors to do 10 times the work. Fewer transistors may be all that’s needed, if none are wasted on functions that aren’t required right now. Transistors would simply be on call, available for whatever was required at the moment.

Programming such a beast would be a whole new challenge. How do you write code for a moving target? According to RC partisans, you don’t. In Elixent’s case, there is no software as we know it. The original program was simply the input function, a way to define what you wanted. The output function was the hardware itself, which has all the program information it needs. Cue *Twilight Zone* music. 📌

Jim Turley is an independent analyst, columnist, and speaker specializing in microprocessors and semiconductor intellectual property. He is the former editor of Microprocessor Report and Embedded Processor Report and host of the annual Microprocessor Forum and Embedded Processor Forum conferences. You may write to him at jim@jimturley.com or visit his web site at www.jimturley.com.

RESOURCES

VAutomation

ARC International
(603) 882-2282
www.vautomation.com

CMOSexod

www.cmosexod.com/freeip.htm

Elixent Ltd.

+44 117 917 5770
Fax: +44 117 917 5779
www.elixent.com

Lexra Inc.

(408) 573-1890
Fax: (408) 573-1898
www.lexra.com

OpenCores

www.opencores.org/projects

SOURCES

ARCTangent, ARCHitect

ARC International
(603) 882-2282
Fax: (603) 882-1587
www.arccores.com

Handel-C

Celoxica
(408) 626-9070
+44 (0) 1235 863 656
Fax: (408) 626-9079
Fax: + 44 (0) 1235 863 648
www.celoxica.com

LEON-1

Distributed by The European Space Agency
+33 1 5369 7654
Fax: +33 1 5369 7560
www.estec.esa.nl/wsmwww/leon/

ST210

Agilent Technologies
(650) 752-5000
http://we.home.agilent.com

STMicroelectronics

(617) 225-2066
Fax: (617) 225-7918
http://us.st.com

Jazz processor

Improv Systems, Inc.
(978) 927-0555
Fax: (978) 927-0999
www.improvsys.com

Xtensa

Tensilica
(408) 986-8000
Fax: (408) 986-8919
www.tensilica.com

The Free-IP Project

XESS, Corp.
(919) 387-0076
Fax: (919) 387-1302
www.free-ip.com
www.xess.com.

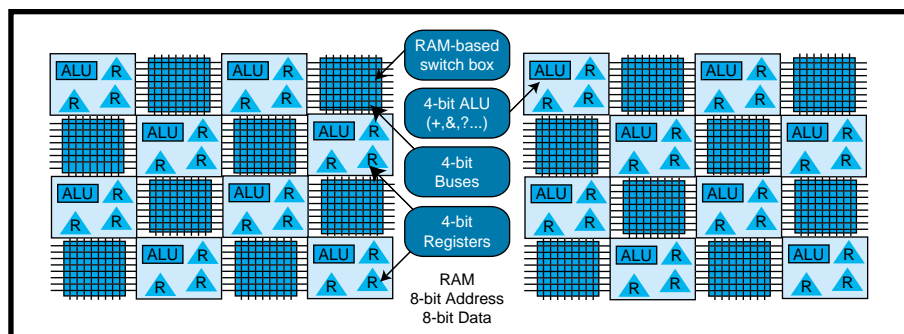


Figure 5—Elixent combines configurable processors with field-programmable logic to create a constantly changing processor. Elixent’s tool switches implementations on the fly based on your workload.