

Turbocharging Your CPU with an FPGA-Programmable Coprocessor

Coprocessor synthesis adds the optimal programmable resources.

by Barry O'Rourke
Application Engineer
CriticalBlue
barry.orourke@criticalblue.com

Richard Taylor
CTO
CriticalBlue
richard.taylor@criticalblue.com

What do you do when the CPU in a board-level system cannot cope with the software upgrades that the next-generation product inevitably requires? You can increase the clock speed, but when – not if – that approach runs out of steam, you must design in a faster CPU, or worse, an additional CPU. Worse yet, you could add another CPU board if the end product's form factor and cost target allow it.

In this article, we'll discuss some of the pros and cons of these approaches and

describe a programmable coprocessor solution that leverages an on-board Xilinx® FPGA to turbocharge the existing CPU and deliver the pros with none of the cons. You do not need processor design expertise, nor will you have to redevelop the software. Moreover, if the board already deploys an FPGA with sufficient spare capacity, the incremental silicon cost of the solution is zero.

The Brute Force Approach

Deploying faster or additional CPUs is an approach that is scalable as long as the software content growth remains relatively linear. However, software content growth is now exponential. A faster CPU might solve the problem temporarily, but it will soon be overwhelmed, necessitating a multiprocessor solution.

Many products cannot bear the redesign and additional silicon costs of a multiproces-

sor solution. Design teams do not often have the time to implement such a solution, and most teams certainly do not have the resources to re-architect the whole multiprocessor ensemble every time the software content exceeds the hardware's capacity to process it. It is more than just a hardware development problem – it is a major software partition and development problem.

When a methodology breaks, you must re-examine the fundamentals. Why can't a general-purpose (GP) CPU deliver more processing power? The answer is that most such CPUs offer a compromise between control functions and the parallel processing capability required by computationally intensive software. That capability is limited by a lack of both instruction-level parallelism and parallel processing resources.

When you deploy an additional general-purpose CPU to assist in the execution of, for example, a 20,000-line MPEG4 algo-

rithm, you add only limited parallel processing capability, along with unnecessary control logic, all wrapped up in a package that consumes valuable board space. It is an expensive, brute force approach that requires the multiprocessor partitioning of software that has generally been developed for only one processor; software redevelopment where the additional CPU's instruction set differs from that of the original CPU; the development of complex caching schemes; and the development of multiprocessor communication protocols – a problem exacerbated by the use of heterogeneous real-time operating systems.

However, even this may not deliver the requisite results. A major Japanese company recently published data that illustrates this point. The company partitioned software developed on one CPU for execution on four CPUs and achieved a maximum acceleration of only 2.83x. A less-than-optimal partition may have failed to utilize the additional parallel processing resources as efficiently as possible, but communications overhead surely played a role as well. In other words, maximizing CPU performance does not necessarily maximize system performance.

The Cascade Coprocessor Synthesis Solution

You can circumvent these problems by synthesizing an FPGA-implemented programmable coprocessor that, acting as an extension of the CPU, supplies the missing parallel processing capability. It increases both instruction-level parallelism and parallel processing resources. A mainstream engineer can design it in a matter of days without having any processor design expertise; CriticalBlue's Cascade synthesizer automatically architects and implements the coprocessor.

Moreover, Cascade optimizes cache architecture and communications overhead to prevent performance from being "lost in traffic." In other words, it boosts not only CPU performance but overall system performance.

The FPGA coprocessor accelerates software offloaded from the main CPU as-is, so no software partitioning and redevelopment are required, although you can also

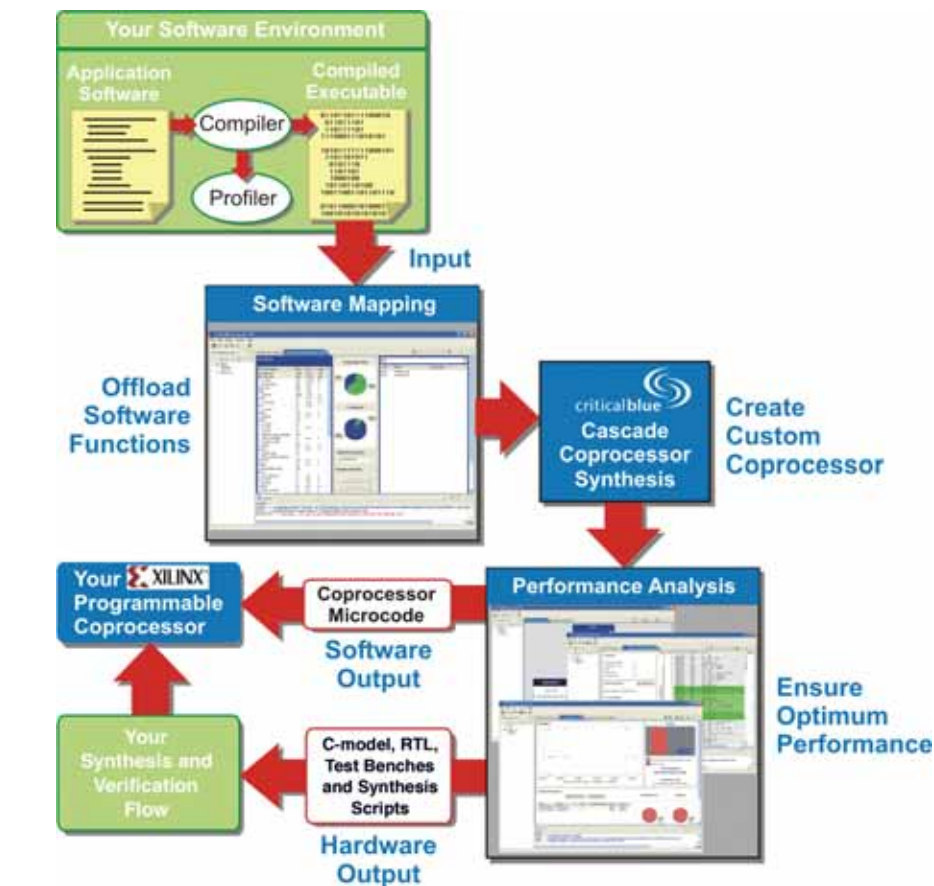


Figure 1 – Cascade coprocessor synthesis flow

co-optimize software and hardware. Cascade thus supports both full legacy software reuse and new software deployment. The former is especially important if you have inherited the legacy software and do not necessarily know how it works.

Unlike fixed-function hardware, a Cascade FPGA coprocessor is not restricted to executing only one algorithm, or part thereof. Cascade can synthesize an FPGA coprocessor that executes two or more disparate algorithms, often with only marginally greater gate counts than a coprocessor optimized to execute only one.

A Cascade FPGA coprocessor is reprogrammable. Although it is optimized to boost the execution of particular algorithms, it will execute other algorithms, too.

Adding a Cascade FPGA coprocessor circumvents the problems of adding a CPU. As you add additional software to the system, it can run on an existing FPGA coprocessor or you can add a new one opti-

mized for that software – all with software developed for single processor operation. Adding multiple coprocessors can be more effective, less costly, and less time consuming than deploying multiple processors.

How Does It Work?

First, you should determine which software routines must be offloaded from the CPU. Cascade analyzes the profiling results of the application software running on the CPU to identify cycle-hungry candidates (Figure 1).

Cascade then automatically analyzes the instruction code – including both control and data dependencies – and maps the selected tasks onto multiple coprocessor architecture candidates that comply with user-defined clock rate and gate count specifications. For each candidate, the tool provides estimates of performance, gate count, and coprocessor/CPU communication overhead. You then select the optimum architecture.

Cascade generates an instruction- and bit-accurate C model of the selected coprocessor architecture for use in performance analysis. Cascade uses the C model together with the CPU's instruction set simulator (ISS) and the stimuli derived from the original software to profile performance, as well as analyze memory access activity and instruction execution traces. The analysis also identifies the instruction and cache "misses" that cause so many performance surprises at system integration. Because Cascade can generate coprocessor architectures so quickly, you can perform multiple "what if" analyses quite rapidly. The model is also used to validate the coprocessor within standard C or SystemC simulation environments.

In the next step, Cascade generates the coprocessor hardware in synthesizable RTL code, which you verify using the same stimuli and expected responses as those used by the CPU. The tool generates the coprocessor/CPU communications circuitry. It also simultaneously generates coprocessor microcode, modifying the original executable code so that function calls are automatically directed to a communications library. The library manages coprocessor handoff and communicates parameters and results between the CPU and the coprocessor. Cascade generates microcode independently of the coprocessor hardware, allowing new microcode to be targeted at an existing coprocessor design.

The Cascade tool provides interactive features if you wish to achieve even higher performance. For instance, it enables you to manually optimize code, co-optimize code and hardware, and deploy custom functional hardware units to achieve greater processing performance. You can explore these manual options in a "what-if" fashion to determine the optimal configuration.

What Are The Results?

The results for single and multiple algorithm execution are shown in Table 1. Executing a single real-time image processing algorithm as-is was 5x faster than on the CPU. Software code optimization and a custom functional unit increased that to 15x.

Application	Single Algorithm		Multiple Algorithms		
Algorithm	Real-Time Image Processing		SHA-1	MD5	SHA-1 + MD5
Lines of Original Code	~200		150	700	850
Code Modified?	No	Yes	No	Yes	Same as MD5
Custom Functional Unit?	No	Yes	No	Yes	Same as MD5
Boost vs. CPU	5x	15x	5x	10x	6.4x
Effort (in Days)	1	3	1	5	1 More Day

Table 1 – Single and multiple algorithm execution

Application	Reprogrammability	
Algorithm	MP3 Decoder	MPEG1 Layer 2 Encoder
Lines of Original Code	~9,700	~16,700
Code Modified?	No	No
Custom Functional Unit?	No	No
Boost vs. CPU on its Own Dedicated Coprocessor	2.13x	1.62x
Boost vs. CPU on the Other's Coprocessor	1.18x	1.19x
Effort (in days)	2	1

Table 2 – Coprocessor reprogrammability with as-is code re-use

In the case of multiple algorithms, a coprocessor synthesized for SHA-1, a secure hash algorithm for cryptography applications, achieved a boost of 5x. A separate coprocessor with a custom functional unit synthesized for MD5, another hash algorithm, achieved 10x. A coprocessor optimized for both achieved 6.4x – using only 8% more gates than either of the two single coprocessors.

The power of reprogrammability is demonstrated in Table 2. One of our customers synthesized a coprocessor to execute an MP3 decoder algorithm, achieving a 2.13x boost despite using unoptimized code. The designer then generated a coprocessor to execute an MPEG1 Layer 2 encoder algorithm, also with no code optimization, achieving a 1.62x boost. When each algorithm was executed on the other's coprocessor, they achieved boosts of 1.18x and 1.19x, respectively. Code optimization would have improved this performance further, and custom functional units to execute serial operations even more.

How Can You Implement It?

You can implement Cascade FPGA coprocessors in multiple Xilinx families: Virtex™-II, Virtex-II Pro/Pro X, Spartan™-3/3E/3L, and Virtex-4 FPGAs.

Cascade's synthesizable RTL output works with Synopsys, Synplicity, and Xilinx synthesis tools and is fully integrated into the Xilinx ISE™ tool flow, while the verification approach works with the Xilinx ISE tool flow and Mentor Graphics's ModelSim XE/PE.

Cascade can target any Xilinx-populated board without translation or modification – no family-specific design kit is required.

Conclusion

If you want to boost the processing power of your design without deploying new or improved microprocessors, and if you want to use software without repartitioning and redevelopment, contact CriticalBlue at info@criticalblue.com. We can tell you how to do it – without having to become a processor designer. 