cadence®

Ten Reasons to Optimize a Processor

By Neil Robinson

SoC designs today require application-specific logic that meets exacting design requirements, yet is flexible enough to adjust to evolving industry standards. Optimizing your processor allows you to meet these design needs while improving data throughput, hardware functionality, and firmware flexibility. The Cadence® Tensilica® Xtensa® customizable processor provides an easy-to-use alternative to hand-coded RTL blocks that allows designers to achieve the same high-computation throughput as RTL hardware accelerators, improving performance and reducing energy consumption while using an automated flow that requires no additional processor verification.

Contents

Introduction

If your design requires application-specific logic to offload the processors to improve performance and reduce energy consumption, you can either create your own register transfer level (RTL) accelerator blocks or consider embedding your logic within the processor to optimize it. Using an optimized processor instead of designing custom RTL blocks gives you a firmware-upgradable implementation with none of the interconnectivity and communications issues associated with an external RTL block. Processor optimization is now highly automated, guaranteeing that the processor IP cannot be broken during the process.

The Cadence Tensilica Xtensa customizable processor can be optimized to meet your application needs as an easy-to-use alternative to hand-coded RTL blocks. Deep pipelines for data, parallel execution units, task-specific state registers, and wide data buses to local and global memories can be added directly into the processor instead of as RTL accelerator blocks, creating a flexible implementation that can be adapted for a specific design. This flexibility allows Xtensa processors to sustain the same high computation throughput as RTL hardware accelerators and still support the same data interfaces. Xtensa processors can be optimized with an automated flow that requires no extra processor verification and keeps the development tools updated with every change.

This white paper provides 10 reasons to optimize a processor.

Reason #1: Future-Proof Your Design

As industry standards evolve, new algorithms must be incorporated into new and existing products. If you hard-code the algorithm in RTL, you cannot make any modifications after the chip is produced. Implementing the design in a processor, however, enables you to make firmware changes after silicon production.

With customizable processors, you can improve efficiency by embedding the logic you were going to put in an external RTL block right into the processor's data path. You can also customize the data path to the exact width required. If you need to process 56-bit data, you can process that all in one chunk rather than two 32-bit operations.

As your processor becomes more efficient, the need to offload so many tasks to dedicated hardwired RTL blocks is greatly reduced if not eliminated. These functions, previously in RTL, are now software programmable in the processor itself and can therefore be easily changed to keep pace with evolving standards.

Reason #2: Avoid Lengthy RTL Verification Time

With standard RISC processors, you build your own RTL blocks for acceleration—which means months of lengthy, complex verification cycles. In most RTL blocks, the finite state machine (FSM), which contains nothing but control logic, causes most of the design and verification risk due to its complexity. A last-minute design change made to an RTL acceleration block is more likely to affect the FSM than the datapath because the FSM contains most of the design complexity.

Customizable processors reproduce the RTL block's ability to have wide datapaths—which can be guaranteed correct-by-construction by the processor vendor—while reducing the risks associated with FSM design because processor-based FSMs are firmware-programmable.

Your single design can simultaneously support multiple standards with the custom operations you create. This means that you can keep pace with new and developing standards by making quick firmware changes avoiding lengthy RTL redesign and verification time.

Datapath elements are controlled differently than their RTL counterparts. The processor-based FSM is implemented in firmware, which greatly reduces the amount of effort needed to fix an algorithm bug or to add new features. In a firmware-controlled FSM, control-flow decisions occur in branches, load and store operations implement memory accesses, and computations become explicit sequences of general-purpose and application-specific instructions.

Many important benefits can be realized by migrating from hand-coded RTL accelerators to microprocessors enhanced with application-specific datapaths controlled by firmware-based FSMs:

- Added flexibility—Simply changing the FSM firmware changes a block's function.
- Software-based development—The ASIC design team can use fast, low-cost software tools to implement most chip features, adding more value to the silicon in less time.
- Faster, more complete system modeling—Even the fastest RTL logic simulator may not exceed a few system-simulation cycles per second, while firmware simulations for processors run at hundreds of thousands or even millions of cycles per second on instruction-set simulators.
- Unification of control and data—No modern system consists solely of hardwired logic—there is always a processor running software. Moving functions that may have been previously implemented with hand-coded RTL functions into a processor removes the purely artificial separation between control and data processing and simplifies the system's design.
- Time-to-market—Using processors simplifies ASIC design, accelerates system modeling, and speeds hardware finalization, which in turn gets the product to market faster. After product introduction, firmware-based FSMs easily accommodate changes to standards because implementation details are not "set in stone."
- Designer productivity—The engineering manpower needed is greatly reduced when compared to developing and verifying custom RTL acceleration hardware. A processor-centric ASIC design approach permits graceful project-schedule recovery when bugs are discovered.

Reason #3: Reduce Energy Consumption

While some people think that all RISC processors deliver about the same performance per clock cycle, designers can get a significant performance improvement for each clock cycle by optimizing the processor for a specific application.

A designer can add custom instructions to the base instruction set architecture (ISA) for the Xtensa customizable processor to significantly reduce the number of cycles required for tasks. As a result, the optimized processor will be slightly larger and have a higher average power dissipation per clock cycle but, as the cycle count reduction is proportionally more than the increase in area, the total energy consumed (power-per-cycle multiplied by total cycles) is substantially reduced.

For example, a 20% increase in power dissipated per clock cycle, offset by a 3X speed up in task execution, actually reduces energy consumption by 60%. This reduction in required task-execution cycles allows the system either to spend more time in a low-power sleep state, or to reduce the processor's clock frequency and operating voltage, leading to further reductions in both dynamic and leakage power.

Reason #4: Get a Unique and Proprietary Processor

When you use standard processor IP in your design, it is easier for someone else to copy because they can use that same standard processor IP in their design. However, when you optimize a processor, even just a little bit, that core is now unique to you. No other company can replicate your version of that processor, protecting your intellectual property by making it harder for competitors to copy your ideas. You get higher performance that consumes less power, and you will have a version of a microprocessor that no one else can buy.

Even the software tool chain for your unique processor is only available to those you provide it to. This means that if someone else gets your processor, they cannot take advantage of anything you added unless you provide them with the tools to do so.

When you customize an Xtensa processor, you automatically get a matching software tool chain that is optimized for your changes. While a generic compiler targeting the base processor might allow your software to run, it will not achieve the levels of performance you can get by using your optimized compiler on your code targeting your processor.

Customization can add special registers (sized to the natural data types of the tasks to be performed) and execution units that perform task-specific algorithms more efficiently, often in one or two clock cycles. This design approach keeps clock rates and energy consumption low, and makes it more difficult for an external observer to figure out what is going on.

You may say that you are not a processor designer and do not know how to add registers and execution units to a processor, but you probably know Verilog. With our automated process, you simply need to write a few lines of Verilog-like code. Our processor generator automatically figures out how to modify the processor to get the results you want.

Reason #5: Use an Automated Process

The process of creating an optimized processor for a specific application is highly automated. Cadence lets you use a two-step automated process, with feedback along the way.

Step 1: Build your processor

Test out different configuration options. It is as easy as checking a box to pick the options for your application. The Xtensa Processor Generator (XPG) allows quick configuration of many Xtensa options using simple click-box screens. You can choose from a wide range of options including basic interfaces, bus widths, debug, trace port, number of interrupts, memory subsystem options, parity or error correcting code, and even advanced options such as including Cadence Tensilica HiFi DSPs for audio/voice, ConnX DSPs for communications, and much more.

Check your work by building the processor using our automated Xtensa Processor Generator, which creates an RTL description of the configured processor and generates tailored versions of all necessary software development tools including the compiler, assembly, debugger, and instruction set simulator (ISS). It also can generate a C or SystemC simulation model of the processor and EDA synthesis scripts. No manual work is required to match software development tools and the processor.

Then run your code on the ISS and see if any areas need improvement. If so, you can move on to step two.

Step 2: Optimize and accelerate

Write Tensilica Instruction Extensions (TIE)—a Verilog-like language—to accelerate the hot spots in your code. It is easier and faster to add new instructions to an Xtensa processor than to design a Verilog hardware block to perform that function. TIE improves performance in your processor design by an order of magnitude. For more information see the white paper TIE—The Fast Path to High Performance Embedded SOC Processing.

You never touch the actual processor RTL. Instead, the TIE instructions are fed into the Xtensa Processor Generator and a complete new RTL description of the optimized processor is automatically created, along with all the matching software development tools. Our automated process ensures your new core will be correct-by-construction—we guarantee it. We have delivered thousands of different designs this way, shipping billions of cores per year.

Adding functions to an Xtensa processor never compromises the underlying base Xtensa instruction set, thereby ensuring availability of a robust ecosystem of third-party development and debug tools. Xtesna processors can run all major operating systems, and always come with a complete, automatically generated software development tool chain including the Eclipse-based Xtensa Xplorer Integrated Development Environment (IDE), a world-class optimizing and vectorizing C/C++ compiler, a cycle-accurate SystemC-compatible simulation model and instruction set simulator, the full industry-standard GNU tool chain, and EDA synthesis scripts.

Reason #6: Avoid the I/O Bandwidth Bottleneck

A processor's main bus represents a significant I/O bottleneck—so much so that processors have lacked the bandwidth required by many tasks performed in SoCs.

To relieve the congestion on the main bus, processors have evolved to provide new ways to support high-performance I/O. Xtensa processors let you achieve data transfer rates that can match those of hand-designed RTL blocks, offering ways to directly connect to various types of data structures without using the main system bus:

- Cache, RAM, and ROM interfaces provide direct connections to local memories.
- Ports act like general-purpose I/O (GPIO) that can be used to directly connect two Xtensa processors or an Xtensa processor to external RTL. Each Port can have up to 1,024 wires, allowing wide data to be transferred easily without the need for multiple load/store operations. Ports are particularly useful to write control updates and read status information.
- Queue interfaces provide a high-speed mechanism to transfer streaming data without memory buffering. Input and output queue interfaces connect to FIFOs and look like traditional processor registers—but without the bandwidth limitations of local and system memory access. Each Queue can make a transfer of up to 1,024 bits every clock cycle. Many interfaces can be added and multiple interfaces can be operated per cycle.
- Memory Lookup interfaces are useful for connecting RAMs as table lookups or for connecting fixed-latency hardware computation units. Memories connected to these lookup interfaces are outside the normal processor memory map and are read and written directly from the processor using their own instructions.

Designers can easily specify all of these features to be added to the Xtensa processor with full support in the system models created by the Xtensa Processor Generator (XPG). The behavior of the interface is automatically reflected in the custom software development tools, instruction set simulator, bus functional model, and EDA scripts. As the processor is automatically created using our patented technology, it is pre-verified and correct-by-construction—no need to re-verify the processor.

Reason #7: Optimize Using C, No Need for Assembly Coding

You no longer have to perform assembly-level code optimization to achieve high performance because the Xtensa C/C++ compiler (XCC) produces an optimum implementation that often yields similar or better performance than hand-optimized assembly in a fraction of the time. The compiler will vectorize (for SIMD) and bundle instruction operations (for FLIX, a form of VLIW with variable slot widths) to gain as much parallelism as possible. Since C is easier to maintain, elimination of assembly coding is a significant advantage. As designs are updated for next generation products, coding in C is an advantage for upward design migration and portability.

Reason #8: Get Better Area/Performance Tradeoffs

Thinking about moving up to a bigger core because you need more processor performance? That bigger processor will not only take more area, but will also consume more power. Instead, you can start with a highly efficient base architecture and just add the performance where you need it. With the Xtensa processor, you can get a lean, mean processing machine, without the overhead of a bigger general-purpose processor.

Reason #9: Make Your Design Team More Productive

Sure, you could design your own processor or DSP, or design your accelerator functions in RTL. But the design cycle usually takes over a year, not including all the time needed to design the matching software tool chain—the compiler, debugger, instruction set simulator, etc. There is also no guarantee that the exact processor specification or RTL design you pick to implement is the best one.

Instead, you can use an Xtensa customizable processor and compare and contrast different alternatives to pick the best architecture. Then, when you have met your performance, area, and power goals, you can trust the proven Cadence automated process of generating a matching complete software tool chain for the exact processor you have designed. Thousands of designs have been completed this way, and we guarantee that the software tools, as well as the processor, are correct-by-construction.

The engineering manpower needed to develop and verify custom RTL acceleration hardware is greatly reduced. A processor-centric ASIC design approach permits graceful software-based project-schedule recovery when a bug is discovered. Even the time-consuming RTL verification cycle is removed—you just need to verify that the processor is doing what you think you asked it to do, rather than face the standard hardwired RTL verification challenges.

Reason #10: Optimize Because You Can

With automated tools, why not try to optimize a processor for your exact requirements? You will be amazed at the range of options you have to get the efficiency you need from a programmable processor-based solution. With our development and profiling tools, you can test out different configurations and options to see what gives you the best combination of area, power, and performance for your exact application.

You no longer need to worry that you are not a processor designer. Even if you have never designed a processor before, if you know what your applications need—what you might have put in an RTL block in old designs—you can use our automated process to customize your processor.

You do not have to go into the processor RTL and make any modifications. As a matter of fact, we do not want you to touch the RTL—that is all created by our Xtensa Processor Generator.

Instead, you use a Verilog-like language, TIE, to describe the features you want to add. Our automated process then generates the processor with your new requirements as well as the matching software tool chain and hardware deliverables.

Additional Information

For more information on the unique abilities and features of Cadence Tensilica Xtensa processors, see ip.cadence.com.

4397 04/15 SC/DM/PDF