cadence®

Reduce Development Risk and Add Programmability with RTL-Like Performance and Connectivity

To effectively use processors for data-intensive functions, designers need a quick, foolproof way to customize those processors for the task at hand. A customizable processor can connect to existing RTL blocks and provide additional computational horsepower tailored to the exact data type needed—with less effort than hand-coding RTL finite state machines or intricate engines. Programmability provides the flexibility to make changes close to and after silicon production. This white paper discusses how Cadence® Tensilica® Xtensa® processors can be optimized for use in the system-on-chip (SoC) dataplane.

Contents

Introduction 1
A Closer Look at RTL Design.....2
Advantages of Using Processors Instead of RTL.....3
Why Are Conventional Processors Used Infrequently in the Dataplane? 3
Xtensa Processors as RTL Alternatives.....4
Achieving the Data Throughput You Need 4
Achieving RTL I/O Performance Using Processor Ports 5
Achieving RTL I/O Performance Using Processor Queues 6
Achieving RTL I/O Performance Using Processor Lookups 7
Achieving the Performance You Need 7
A Formal Structured Processor Creation Language: TIE 7
Automated—Easy to Add Ports, Queues, and Lookups.....8
Hundreds of Checkbox Functions to Choose From.....8
You Do Not Need to Be a Compiler Expert.....8
Conclusion—Use Xtensa Processors in the Dataplane.....9
Additional Information 9

Introduction

The main control processors are good at executing a wide range of algorithms, but depending on application, designers often need a lot more performance for major functions such as audio, video, baseband, security, protocol processing, and much more. These functions comprise the dataplane of the SoC.

While designers understand how to use a processor to perform the control functions in a SoC design, conventional control processors cannot handle many of the data-intensive functions. Historically, designers have created RTL accelerator blocks for these challenging functions. However, RTL blocks take a long time to design and verify, and are not programmable—and thus not flexible enough to easily handle multiple standards or post-tapeout design changes.

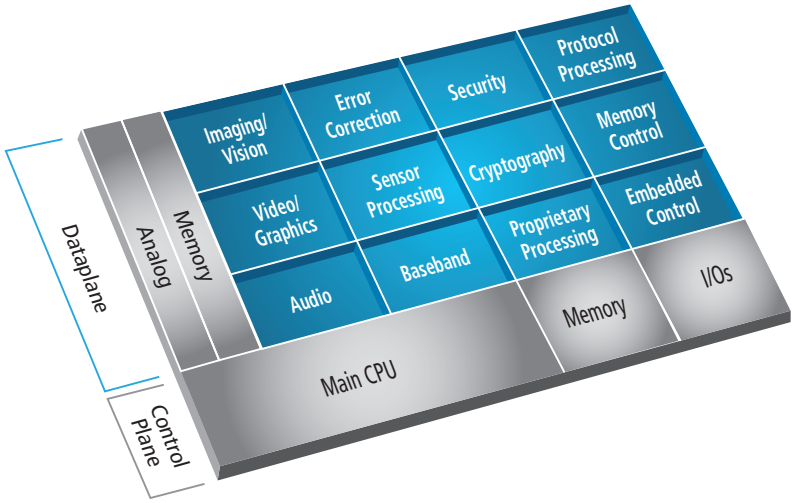

Figure 1: The Dataplane: The Largest Part of the SoC Design

The dataplane has become the largest part of a SoC design (see Figure 1), and it certainly takes the longest to design and verify. In most companies, a system architect divides the dataplane tasks into separate teams of engineers, each taking months to design and verify their part of the SoC. Meanwhile, the software team tries to develop software, even though the hardware is not yet designed—a huge challenge that often requires significant re-work once the hardware design is completed.

HDLs, such as Verilog or VHDL, are traditionally used to hand-design digital functions in the dataplane, including functions tightly coupled to conventional CPUs, often referred to as hardware acceleration blocks or hardware accelerators. Designing this dataplane hardware in RTL takes significant design time, not just for capturing the design intent, but for verification of the new hardware. In fact, verification can consume as much as 70% to 80% of the total hardware design time.

The two major problems with RTL design are:

- It takes too long to design and verify.
- The resulting hardware is not programmable. All functions are hard-wired and can't be easily reprogrammed or changed after the silicon is produced, so most changes require a silicon re-spin.

A Closer Look at RTL Design

Before we can look at alternatives to RTL design, it is important to take a closer look at a typical RTL block.

Figure 2 shows the internal structure of a generic RTL accelerator block. The block's datapath appears on the left and its state machine appears on the right.

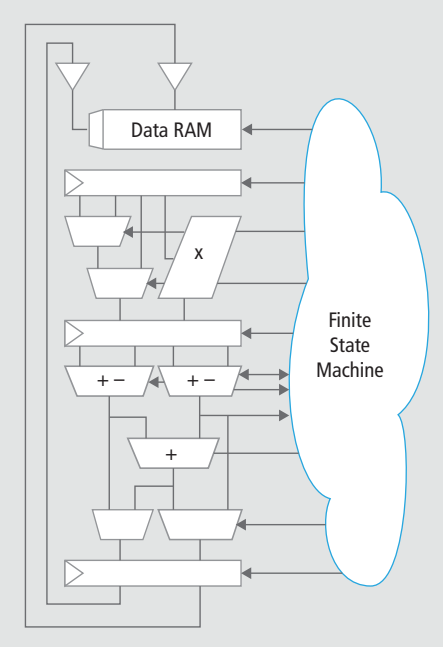

Figure 2: Hardwired RTL = Datapath Plus State Machine

In most RTL accelerators, the datapath consumes the vast majority of the gates. A typical datapath may be as narrow as 16 or 20 bits or range up to hundreds of bits wide, depending on the target task or application. RTL datapaths typically contain many data registers and often include significant blocks of RAM or interfaces to blocks of memory that are shared with other RTL blocks.

By contrast, the RTL logic block's finite state machine (FSM) contains nothing but the control logic. All the nuances of sequencing data through the datapath, all the exception and error conditions, and all the handshakes with other blocks are captured in the FSM. While the FSM is often small in area compared to the datapath, it most often embodies most or all of the design and verification risk due to its logical complexity.

A late 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. And this is where the biggest problems occur with RTL blocks—in the FSM.

The inherent advantage of using RTL design for computationally intensive design elements is that the datapath can be tailored to exactly the native data type the function requires. For example, audio sample data can be 24 bits of precision, 56-bit values can be manipulated in a DES security block, or varying widths of data can be computed in a communications baseband design. Conventional control CPUs or fixed-function DSPs by contrast are limited to only 16-bit or 32-bit data types—any mismatch between the real-world precision and the pre-ordained internal values of the processor leads to dramatic performance losses.

What is the alternative to dedicated RTL blocks? Using a custom processor for embedding the datapath logic and gates, and embedding the FSM control logic into the processor's firmware to decouple the datapath and the FSM control logic. This separation minimizes hardware design changes and provides flexibility to keep pace with updates that can easily be made in firmware. Custom processors enable the RTL block to have wide, non-integer sized 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.

Advantages of Using Processors Instead of RTL

Processors enhanced with application-specific datapaths controlled by firmware-based FSMs have many advantages over hand-coded RTL accelerators:

- **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 processor simulations 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 for processors 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.
- **Reduced risk**—Customized processors reduce the risks associated with complex FSM development by replacing hard-to-design, hard-to-verify, hardware FSMs with pre-designed, pre-verified processors.
- **Eliminate assembly coding**— Evolving to a more processor-centric design methodology moves much of the design definition to the C and C++programming languages instead of assembly coding. Application tasks defined in firmware that run on processors can be optimized, through customizable architectural extensions, to match the application performance of custom RTL accelerator blocks.

Why Are Conventional Processors Used Infrequently in the Dataplane?

Most designers chose not to use conventional CPUs and DSPs in the SoC dataplane for four major reasons:

- **Data throughput**—Conventional processors use bus interfaces to transfer data and require clock cycle(s) to store and fetch data from memory, slowing data transfers.
- **Processing speed**—Conventional processors do not provide the data types and ALU functions required to perform data-intensive, real-world computing, so designers turn to RTL.
- **Customization challenges**—Most designers are not processor experts and are hesitant to customize a conventional processor.
- **Customization limitations**—Most processors cannot be customized beyond simple parameterized functions such as memories and interfaces.

Xtensa-based processors uniquely overcome these objections—making them ideal for the SoC datapath.

Xtensa Processors as RTL Alternatives

You can use an Xtensa processor as an alternative to hand-coded RTL blocks by adding the same datapath elements to it as implemented in RTL accelerator blocks. These datapath elements include deep pipelines, parallel execution units, task-specific state registers, and wide data buses to local and global memories. This allows the processor to sustain the same high computation throughput and support the same data interfaces as RTL hardware accelerators. You can optimize the Xtensa processor to run your application more efficiently.

Processor datapaths are controlled differently from their RTL counterparts. Control of a processor's datapaths is not frozen in a hardware FSM's state transitions. Instead, the processor-based FSM is implemented in firmware (see Figure 3), 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.

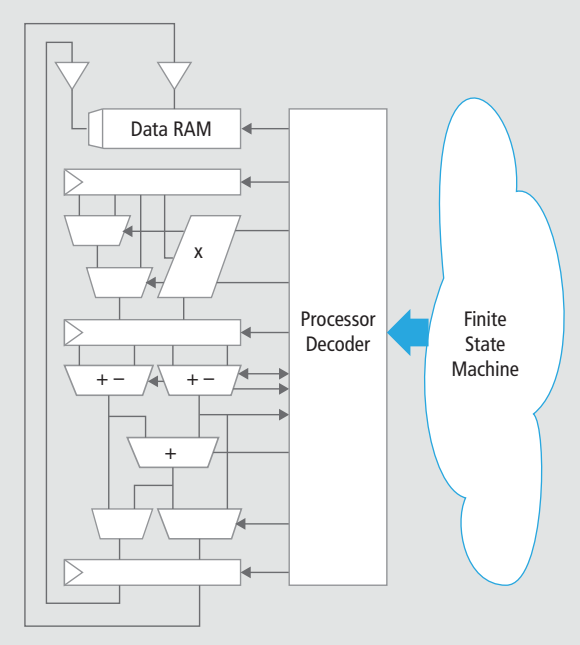

Figure 3: Programmable Hardware Function: Datapath + Processor + Software

Achieving the Data Throughput You Need

Virtually every 32-bit processor on the market uses bus interfaces to transfer data and requires multiple clock cycles to store and fetch data from memory—except Xtensa-based processors, which let you bypass the bus entirely.

SoC buses are increasingly inadequate for high-throughput applications such as video compression/decompression or high-speed networking. Xtensa-based processors increase your flexibility, enabling you to create multiple custom Ports (GPIO), Queue (FIFO) interfaces, and memory Lookup interfaces to provide as much bandwidth as any system can possibly use, virtually unlimited I/O. See Figure 4. This is much like the I/O capabilities of RTL design.

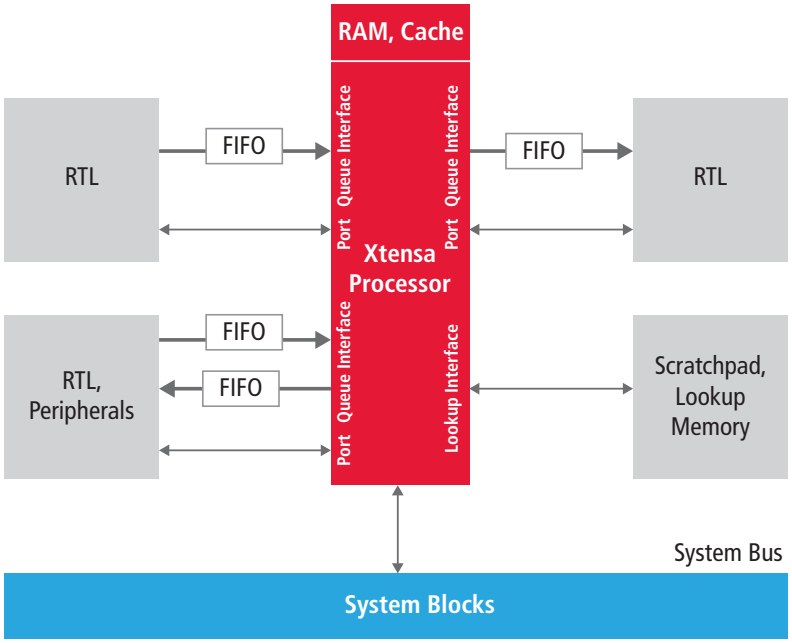

Figure 4: Processor-Based System with Separate I/O Interfaces

The processor Port and Queue interfaces directly connect to external RTL and FIFO blocks and operate in a single cycle in and single cycle out of the processor, giving deterministic performance. The Lookup interfaces directly connect to memories or other system blocks, and are set to a fixed-latency appropriate for that connection. These interfaces can be read and written directly from the processor datapath without using load and store instructions.

These interfaces are used in parallel to perform transactions independently of the local memory interfaces—they are not mapped into the processor's main memory address space.

In addition, the Xtensa processor's Flexible Length Instruction eXension (FLIX) technology—a multi-issue operation technology using a form of VLIW with variable slot widths and without the code bloat issue normally associated with VLIW—allows designers to add separate, parallel execution units to handle multiple simultaneous computational operations.

See the *Increasing Processor Computational Performance with More Instruction Parallelism* white paper for additional information.

Achieving RTL I/O Performance Using Processor Ports

You can directly connect two Xtensa processors or an Xtensa processor to a block of RTL through Xtensa processor Ports. These general-purpose I/O (GPIO) connections allow bandwidth of up to 1,024 bits per cycle on each interface. Ports are often used to control an external logic block or read state and control information through direct wires.

You can add hundreds of Ports, each with as many as 1,024 pins, which boosts maximum I/O bandwidth more than 1,000X relative to conventional 32- or 64-bit processor buses. While it is unlikely any design will utilize that much bandwidth in a single processor, the practical result of this flexibility is to completely remove I/O bandwidth as a limitation to using the Xtensa processor in a particular function.

For example, Figure 5 uses six Ports to connect three pairs of input/output devices to three custom execution units in the processor.

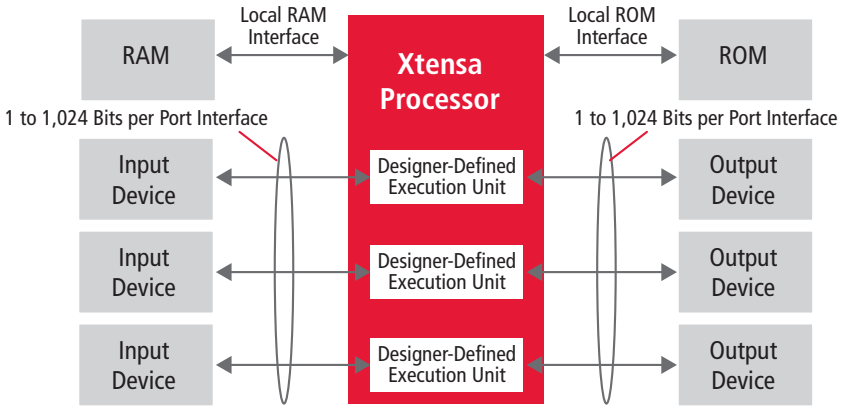

Figure 5: Xtensa Processor with Multiple I/O Interfaces and Execution Units

Note that similar interfaces can be used to attach an existing block of custom RTL accelerator logic to a processor without reducing the I/O bandwidth available to other devices. Figure 6 shows how such an accelerator might be connected to a customizable processor.

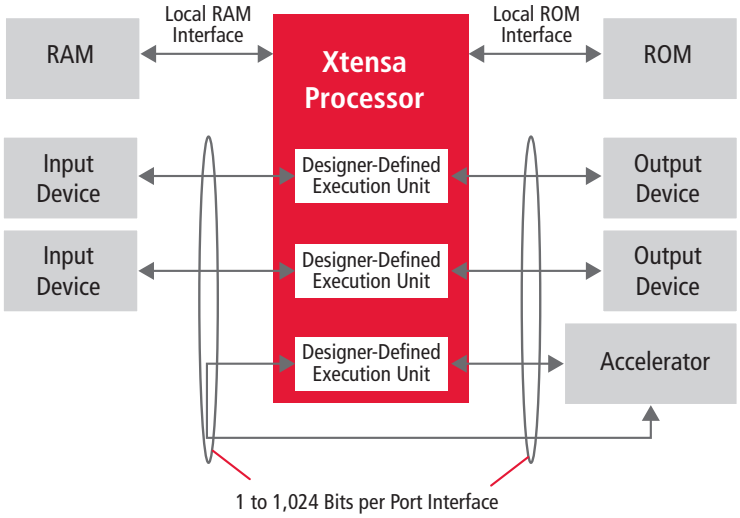

Figure 6: Using Ports to Attach an Existing Acceleration Unit to a Processor

See the *High-Speed Alternatives for Inter-Processor Communications on SoCs* white paper for additional details.

Achieving RTL I/O Performance Using Processor Queues

While Ports are ideal to quickly convey control and status information, the Xtensa processor also allows you to easily add Queues (FIFO interfaces), as shown in Figure 4 earlier. Queues provide a high-speed mechanism to transfer data between the processor and other parts of the system that may be producing and consuming the data at different rates. Input Queues and output Queues operate to the programmer's viewpoint like conventional processor register accesses—but without the bandwidth limitations of local and system memory accesses.

Queues can sustain data rates as high as one transfer every clock cycle. Custom instructions can perform multiple Queue operations per cycle, perhaps combining inputs from two input Queues with local data and sending the computed values to two output Queues. The high bandwidth and low control overhead of Queues allows the Xtensa processor to be used in applications with extreme data rates.

TIE input Queues present a familiar pop/empty/data interface to the external logic, and TIE output Queues present a similar push/full/data interface.

See the *High-Speed Alternatives for Inter-Processor Communications on SoCs* white paper for additional details.

Achieving RTL I/O Performance Using Processor Lookups

Figure 4, shown earlier, illustrates a direct connection to memory via an Xtensa processor Lookup interface to get fast data throughput. Multiple memories can be connected via separate physical Lookup interfaces on the processor, and a single TIE instruction can perform a transaction on all interfaces at once.

The Lookup interface can be used in parallel to perform transactions independently of the local memory interfaces. These directly connected memories are not mapped into the processor's main memory address space.

For example, the Lookup interface can be used to send a request (address + data) to an external device and then receive a response (data) within a fixed number of cycles. The request and response are treated as an atomic transaction within the TIE instruction.

Achieving the Performance You Need

General-purpose processors do not provide the performance required to perform data-intensive functions. On the other hand, the Xtensa customizable processor provides substantially more performance than traditional CPUs and DSPs—performance that can often approach that of custom RTL blocks. The Xtensa processor delivers this high performance in two ways:

- Very fast, direct data transfer (described above)
- Execution unit customizations that dramatically improve processing speed

An Xtensa processor can implement wide, parallel, and complex datapath operations that closely match those used in custom RTL hardware. The equivalent datapaths are implemented by augmenting the base processor's integer pipeline with additional execution units, registers, and other functions developed by the chip architect for a target application.

Many data processing algorithms operate in a loop and need cycle-by-cycle loading and computation for a defined period. The Xtensa processor with a custom execution unit can execute instructions inside a zero overhead loop giving this cycle-by-cycle loading and execution, the same as an RTL block computation.

The result is a new processor with integrated, native instructions and execution units—not a processor with bolted-on coprocessors. The new instructions and registers are available to the firmware programmer via the same compiler and assembler that target the base processor's instructions and register set. Because the instruction extensions greatly accelerate the processor's performance on the targeted algorithm, FSM firmware can usually be written in a high-level language, such as C or C++, to achieve optimum performance and does not need to be coded in assembly.

A Formal Structured Processor Creation Language: TIE

The Tensilica Instruction Extension (TIE) language allows you to specify the customizations you would like to make to an Xtensa processor. This formal structured processor creation language is a Verilog-like language used to describe custom instructions, execution units, I/Os, and processor state. The functionality of custom execution units is described in TIE using combinatorial Verilog.

Virtually any form of computation that you could implement in a datapath can be implemented in a TIE function semantic block. You can also specify with a single TIE language directive that a complex instruction be implemented as a multi-cycle instruction and the TIE Compiler will automatically partition the logic across multiple pipeline stages and adjust the C compiler accordingly. Register files and programmer-visible processor state variables are declared with specialized structured TIE code.

Because TIE is a formal processor description language, it frees you from worrying about integrating your new function into the processor pipeline. You only need to concentrate on the desired computational function—the TIE language and TIE compiler automatically handle the rest. As the TIE language leverages Verilog, it is easy for a RTL developer to quickly learn how to use the language to create powerful processor extensions.

See the *TIE—The Fast Path to High Performance Embedded SoC Processing* white paper for additional details.

Automated—Easy to Add Ports, Queues, and Lookups

Add new Ports, Queue, and Lookup interfaces with simple one-line declarations or checkbox configuration options. The interfaces that you specify are automatically added to Xtensa processors and are 100% modeled in the tools. The full behavior of the interface, like any other modification made to the Xtensa processor, is automatically reflected in the custom software development tools, instruction set simulator, bus functional model and EDA scripts—in about an hour. As it is automated using our patented technology, it is pre-verified and correct-by-construction—no need to re-verify the processor.

Hundreds of Checkbox Functions to Choose From

Optimizing an Xtensa processor for a particular dataplane function does not mean that you reinvent the wheel each time. Cadence provides the designer a series of ready-made options for tuning a processor to meet the application. A designer can fine-tune low-level items like interrupt structures and local memory subsystems, or choose from high-level functional blocks such as a range of fixed-point and floating-point DSPs including our market-leading Tensilica HiFi DSPs for Audio, Voice, and Speech.

Figure 7 shows a sample screen shot of one configuration menu from our Xtensa Xplorer graphical user interface.

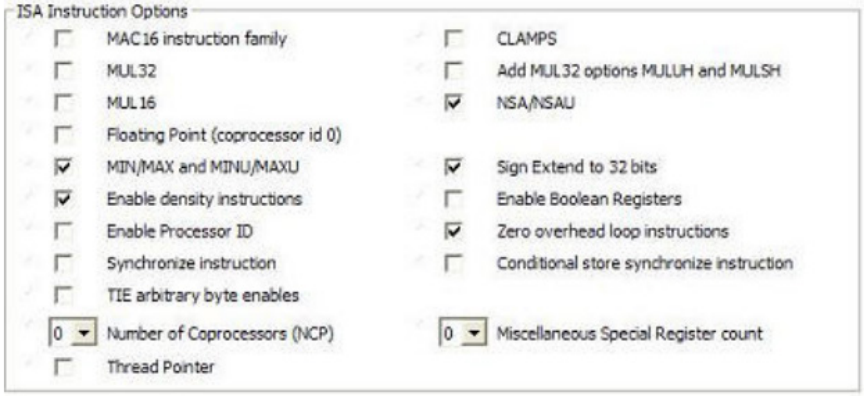

Figure 7: Sample Configuration Window.

You Do Not Need to Be a Compiler Expert

You do not need to be a software or compiler expert to modify an Xtensa processor. We have automated the software tools process. Once you have a specification of the extensions you want implemented in a tailored processor, the Xtensa processor tool chain automates the process of implementing not only the RTL, but also the full suite of software development tools, system models, and EDA scripts. This automation allows Cadence to guarantee the results—they are correct-by-construction.

Adding functions to an Xtensa processor never compromises the underlying base Xtensa instruction set, thereby ensuring a robust ecosystem of third-party application software and development tools is available. Customized Xtensa processors are compatible with leading embedded real-time operating systems, development, and debug tools, and always come with a complete, automatically generated software-development tool chain including: an advanced integrated development environment (IDE) based on the ECLIPSE framework, a world-class optimizing, vectorizing compiler, a cycle-accurate, SystemC-compatible simulation model and instruction set simulator, the full industry-standard GNU tool chain, and EDA synthesis scripts.

Conclusion—Use Xtensa Processors in the Dataplane

The reasons to use a customizable processor in the dataplane of your SoC are clear—flexibility and design productivity. The unique capabilities of Xtensa processor deliver those advantages while also removing the disadvantages typically associated with fixed-function processors and DSPs. It is much faster to customize an Xtensa processor than to design and verify a hard-wired RTL block that cannot change easily for new standards or software—and you do not have to sacrifice performance, area, or energy consumption. Xtensa processors future-proof your designs, letting you make changes in firmware after the silicon is produced.

Additional Information

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