

## Using Processors in the SOC Dataplane

*Designers have long understood how to use a single processor for the control functions in an SOC design. However, there are a lot of data-intensive functions that conventional control processors (CPUs) cannot handle. That's why designers have, for a long time now, designed RTL blocks for these challenging functions. However, RTL blocks take a long time to design and verify. Plus they are not programmable and thus not flexible enough to easily handle multiple standards or post-tapeout design changes.*

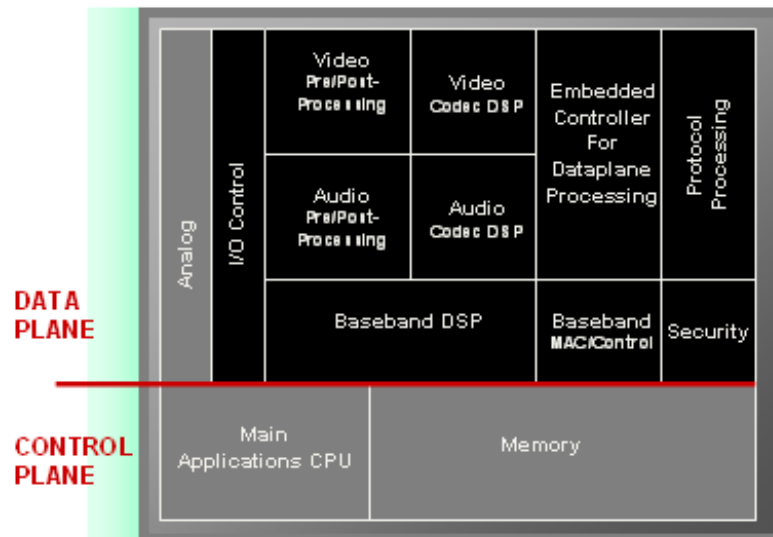
*To effectively use processors in the dataplane, designers need a quick, fool-proof way to customize those processors for the exact task at hand. A dataplane processor (or "DPU") naturally connects to existing RTL blocks and provides additional computational horsepower tailored to the exact data type needed – all this with less effort than hand-coding RTL finite state machines or microcoded engines. With a DPU's programmability, designers now have the flexibility to make changes close to and after silicon production. This white paper explains how DPUs can be effectively used in the SOC dataplane.*

### **The SOC Dataplane Challenges**

By the mid 1990s, IC manufacturing technology had advanced enough to allow the inclusion of a processor in the standard-cell ASIC along with memory and glue logic. This was the start of the SOC era.

Moore's Law has allowed designers to add more functions to each chip. The main control processors were (and still are) good at executing a wide range of algorithms, but designers often need a lot more performance for major functions such as audio, video, baseband DSP, security, protocol processing, and much more (the list depends on the application). These functions comprise the dataplane of the SOC.

Now the dataplane has become the largest part of an 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 start developing software, but because the hardware isn't designed yet, that's a huge software challenge that often requires significant re-work once the hardware design is completed.



**Figure 1. The Dataplane is the Largest Part of the SOC Design**

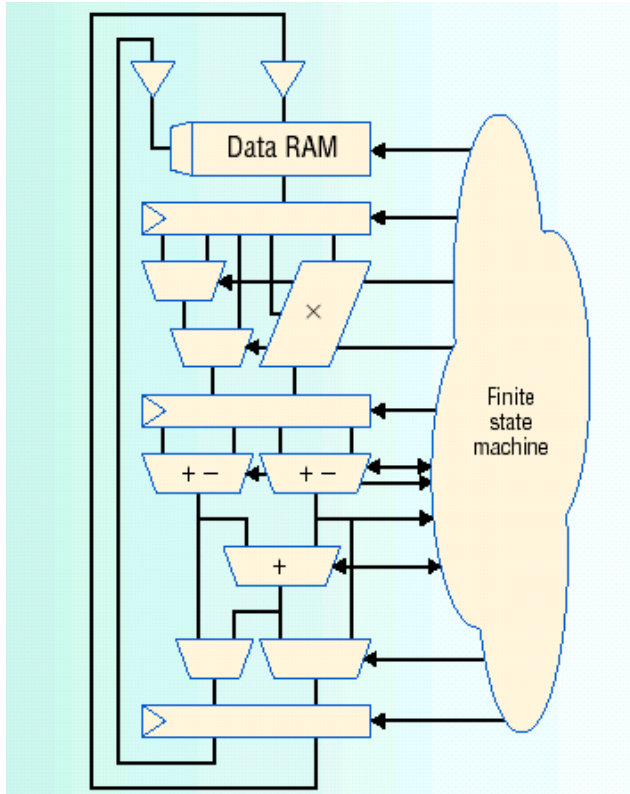
Verilog or VHDL is traditionally used to hand-design digital functions in the dataplane, including functions tightly coupled to conventional CPUs, often referred to as hardware acceleration blocks, or hardware accelerators. Designing this dataplane hardware in RTL takes significant design time, not just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

There are two major problems with RTL design:

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

### A Closer Look at RTL Design

Before we can look at alternatives to RTL design, it's important to take a closer look at a typical RTL block. Figure 2 shows the internal structure of a generic RTL block found in an SOC dataplane. 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 many RTL subsystems, the datapath consumes the vast majority of the gates. A typical datapath may be as narrow as 16 or 32 bits or can be very wide - 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.

The inherent advantage of using RTL design for computationally intensive design elements is that the datapath can be tailored to exactly the kind of native datatype 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.

The RTL logic block's finite state machine (FSM) contains control details. 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 block is much 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.

### Implementing the FSM in a Dataplane Processor

Customizable dataplane processors let designers implement both the datapath and, most importantly, the FSM within the DPU itself for maximum performance and efficiency. Customizable DPUs let designers add the same datapath elements inside the processor core execution units as traditionally implemented in RTL accelerator blocks. These datapath configurations include deep pipelines, parallel execution units, task-specific state registers, non-integer data widths, and wide data buses to local and global memories. This broad configurability allows configured DPUs to sustain the same high computation throughput and support the same data interfaces as RTL hardware accelerators.

However, control of DPU datapaths is very different from the RTL counterparts. Cycle-by-cycle control of a DPU'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.

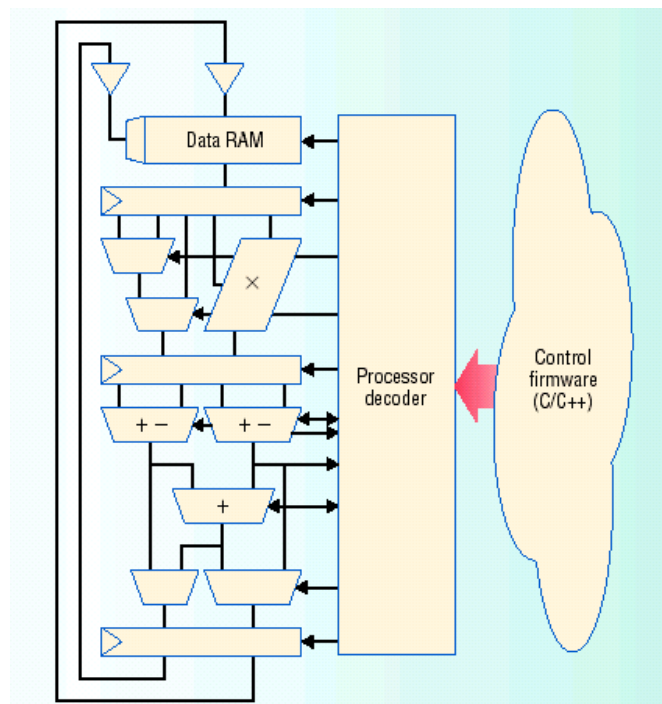


Figure 3. Programmable hardware function: datapath + processor + software

### **Advantages of Using a DPU Instead of RTL**

There are many important benefits to be realized by migrating from hand-coded RTL to DPU cores enhanced with task-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 many chip features and add more value to the silicon.
- **Faster, more complete system modeling:** For a 10-megagate design, even the fastest Verilog logic simulator may not exceed a few system-simulation cycles per second. By contrast, firmware simulations for configured DPUs 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's 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:** As discussed in the above points, using customizable processors simplifies ASIC design, accelerates system modeling, and speeds hardware finalization. These benefits get the product to market

The benefits of being able to make design changes in firmware rather than hardware using a DPU-centric approach cannot be understated. Customized DPUs reduce the risks associated with complex FSM development by replacing hard-to-design, hard-to-verify, hardware FSMs with pre-designed, pre-verified, configurable processor cores and application firmware written in C (or assembly code, if your firmware design team insists).

### **Why Haven't Conventional Processors Been Used More in the Dataplane?**

Most designers haven't used conventional CPUs and DSPs in the SOC dataplane for four major reasons:

1. **Data throughput** – Conventional processor cores use bus interfaces to transfer data and require clock cycle(s) to store and fetch data from memory, slowing data transfers.
2. **Processing speed** – Traditional processors don't provide the data types and ALU functions required to perform data-intensive, real-world computing, so designers turn to RTL.
3. **Customization challenges** – Most designers are not processor experts and are hesitant to customize a conventional processor.

4. Customization limitations – Most processors can't be customized beyond simple parameterizable functions such as memories and interfaces

The next sections show how Tensilica's customizable processors uniquely overcome these objections – making them ideal for the SOC datapath.

### Getting the Data Throughput You Need

Virtually every 32-bit processor core on the market uses bus interfaces to transfer data and requires clock cycle(s) to store and fetch data from memory – except Tensilica's customizable DPU cores. Tensilica is the only company that has developed technology that lets you bypass the bus entirely.

SOC buses are increasingly inadequate for high-throughput applications such as video compression/decompression or high-speed networking. Tensilica's unique feature called "ports and queues," based on tried-and-true system technology, provides the ASIC and SOC design team with as much bandwidth as any system can possibly use, essentially unlimited I/O bandwidth. This is much like the I/O capabilities of RTL design.

For example, the ports and queues features of Tensilica's Xtensa processors allows ASIC and SOC designers to add multiple I/O interfaces to a DPU core as shown in Figure 4. Designers can add as many as 1024 ports to the core and each port can have as many as 1024 pins, which boosts maximum I/O bandwidth more than 1000x relative to a conventional 32- or 64-bit processor buses. While it is unlikely any design will ever utilize that much bandwidth in a single core, the practical result of this flexibility is to completely remove I/O bandwidth as a limitation to the use of the DPU in a particular function.

The ports and queues function as a single cycle in and single cycle out of the DPU core, giving deterministic performance.

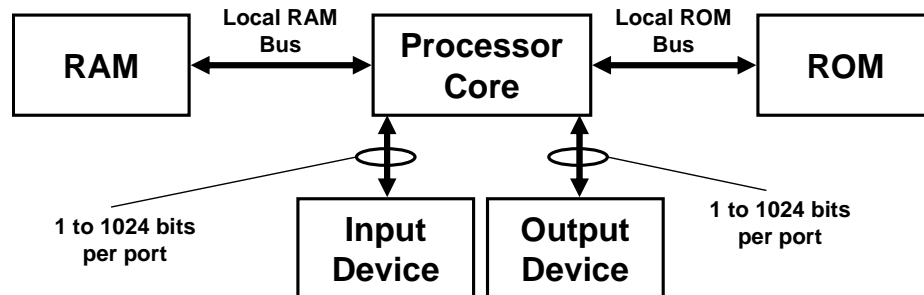
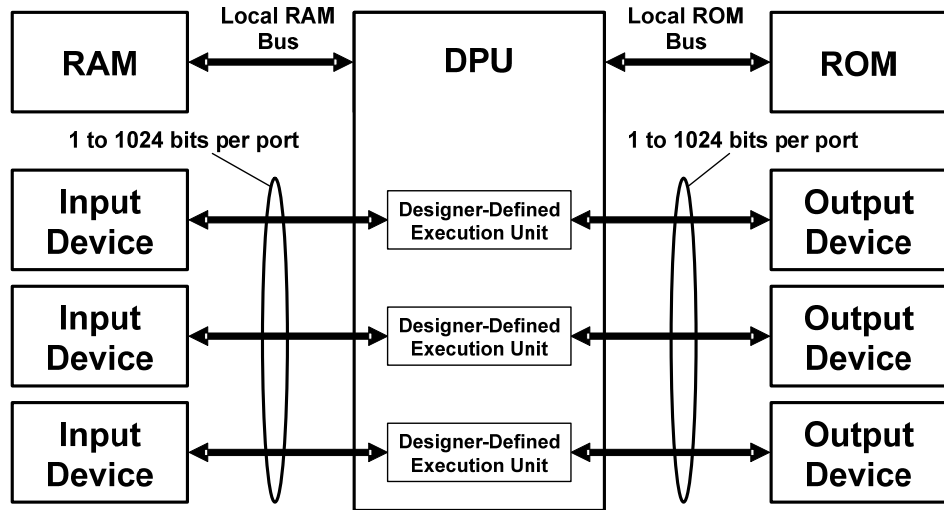


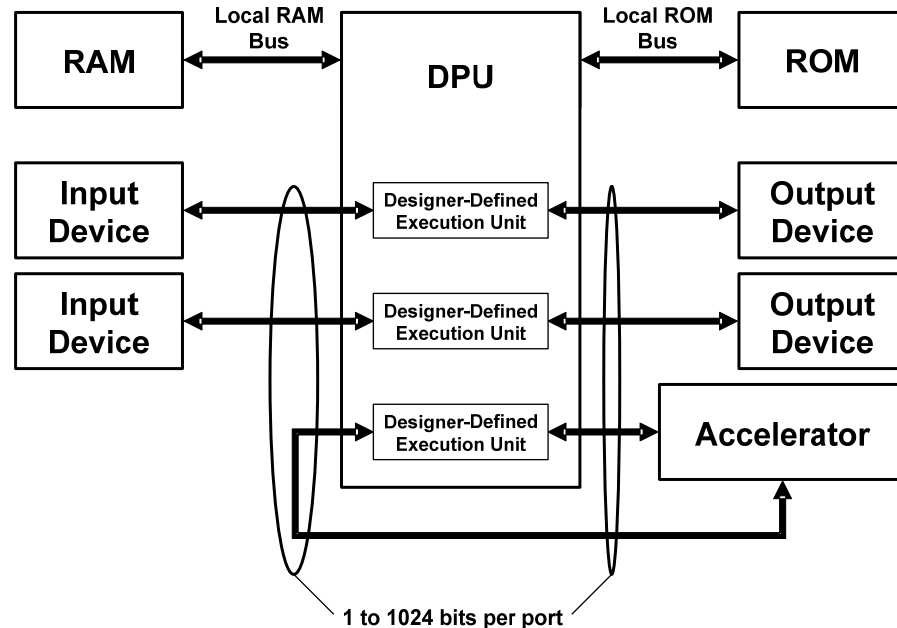
Figure 4: Processor-based system with separate input and output device buses

In addition, Tensilica allows designers to add separate, parallel execution units to handle multiple simultaneous computational tasks. Using both of these features, the resulting configured processor might look something like the block diagram shown in Figure 5, which uses six ports to connect three pairs of input/output devices to three custom execution units in the processor.



**Figure 5. DPU core with multiple I/O ports and execution units**

Note that the same sort of ports 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 TIE ports to attach an existing acceleration unit to a DPU core**

### Getting the Performance You Need

General-purpose processors don't provide the performance required to perform data-intensive functions. Tensilica's DPUs provide substantially more performance than traditional CPUs and DSPs – performance that can often approach that of custom RTL blocks. DPUs deliver this high performance in two ways:

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

A configurable DPU 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. A configurable DPU with a designer-defined execution unit can execute instructions inside a zero overhead loop giving this cycle by cycle loading and execution, the same as RTL block computation.

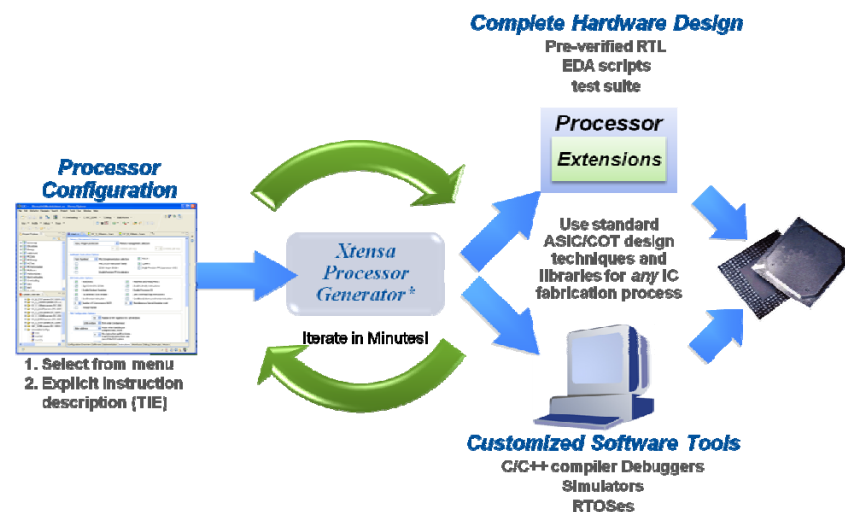
The result is a new processor with integrated, native instructions and execution units - not a processor with bolted-on coprocessors. The new instructions and registers are available to the firmware programmer via the same compiler and assembler that target the foundation processor's instructions and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

### You Don't Need to Be a Processor Expert

You don't need to be a processor expert to modify a Tensilica DPU. We've automated the process. Just start with your C/C++ algorithm or reference specification. Customization of a DPU can be performed by simple designer insight for choosing existing ready-made options or creating new instruction optimizations manually and then using the analysis tools to review performance, size and power consumption. New instructions are written in a simple Verilog-like language, called TIE (Tensilica Instruction Extension)

Once you have a specification of the extensions you want implemented in a tailored DPU, Tensilica's tool chain automates the process of implementing not only the RTL for the core, but also the full suite of software development tools, system models, and EDA scripts. This automation allows Tensilica to guarantee the results; they are correct by construction.

Figure 7 shows Tensilica's automated processor design process.



\* US Patent: 6,477,697

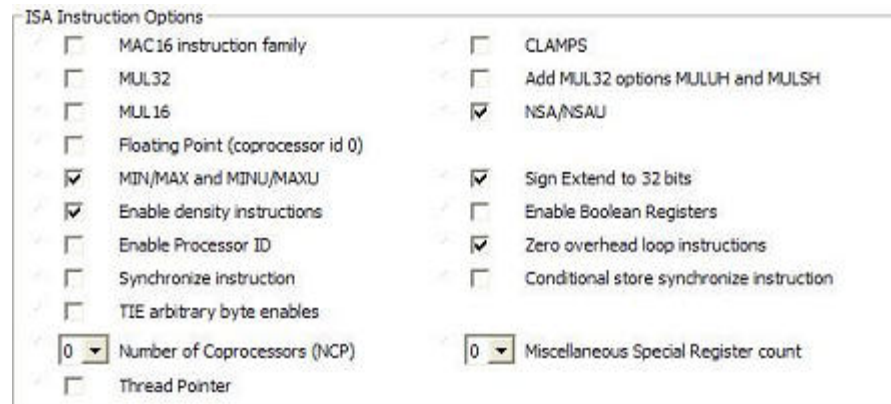
Figure 7. Tensilica's Customizable Processor Design Process

Adding functions to a Tensilica DPU core never compromises the underlying base Xtensa instruction set, thereby ensuring availability of a robust ecosystem of third party application software and development tools. Customizable Xtensa DPUs are always compatible with leading embedded real-time operating systems, debug probes and ICE solutions; and always come with a complete, automatically generated software-development tool chain including an advanced integrated development environment 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.

### Hundreds of Click-Button Functions to Choose From

Optimizing a DPU for a particular dataplane function doesn't mean that you reinvent the wheel each time. Tensilica provides the designer a series of ready-made options for tuning a core 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 range of fixed-point and floating-point DSPs including our market-leading HiFi Audio engines.

Figure 9 shows a sample screen shot of one configuration menu from our Xtensa Xplorer graphical user interface



**Figure 9. Sample Configuration Window.**

There are a lot more options – see our web site for full details.

### A Formal Structured Processor Creation Language: Tensilica's TIE Language

The Tensilica Instruction Extension (TIE) language allows designers to specify the customizations they'd like to make to a Tensilica Xtensa DPU. This formal structured processor creation language is a Verilog-like language used to describe

custom instructions, execution units, I/Os and processor state. Functionality of custom execution units are described in TIE using combinatorial Verilog.

Virtually any form of computation that you could implement in a Verilog datapath can be implemented in a TIE function semantic block. Designers 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 the designer from worrying about integrating his or her new function into the processor pipeline. The only thing a designer needs to concentrate on is the desired computational function – the TIE language and TIE compiler automatically handle the rest. And because TIE leverages Verilog, it is very easy for any RTL developer to quickly learn how to create powerful DPU extensions using the TIE language.

See our web site for examples of TIE instructions. We also have application notes on our web sites that go into more detail and provide actual examples of TIE specifications.

### **Conclusion – Use DPUs in the Dataplane**

The reasons to use a programmable processor in the dataplane of your SOC are clear – flexibility and design productivity. Tensilica unique dataplane processor technology delivers those advantages while also removing the disadvantages typically associated with fixed-function processors and DSPs. Thus it is now much faster to customize a Tensilica DPU – without sacrificing performance or low-power - than to design and verify a hard-wired RTL block that can't change for new standards or software. DPUs let you future-proof your designs, letting you make changes in firmware AFTER the silicon is produced.

Try it in your next design.