

Softening Hardware: Using Application-Specific Processors to Optimize Modern SoC Designs

Authors

Markus Willems

Product Marketing Manager,
Synopsys

Steve Cox

Business Development
Manager, Synopsys

Introduction

Over the past decade, the trend in system-on-chip (SoC) design has been to add more functionality into programmable logic. There are several reasons for this, including (a) software is easier and faster to fix and update, (b) evolving trends and not-yet fully specified standards require flexibility since the final functionality might not be known at the time the hardware design must be locked down, and (c) the desire to reuse SoCs for different products and derivatives, improving the return on investment (ROI) for a single design.

Moving functionality from hardware into software comes at a cost, however. Software requires a processor, which, if not designed for optimal efficiency, could be slower and use more power than dedicated hardware. It often makes sense to implement smaller, specialized processors to tackle specific tasks with highly targeted software instead of one processor that has to run any of 100 possible workloads. For example, in data center applications we are seeing dedicated processors for image recognition, SQL, and machine learning acceleration, to name a few. Mobile SoCs are another prominent example, as they deploy a wide range of specialized processors (some of them referred to as "engines" or "accelerators" to contrast them from standard processors), each tailored to specific tasks, as visible in Figure 1.

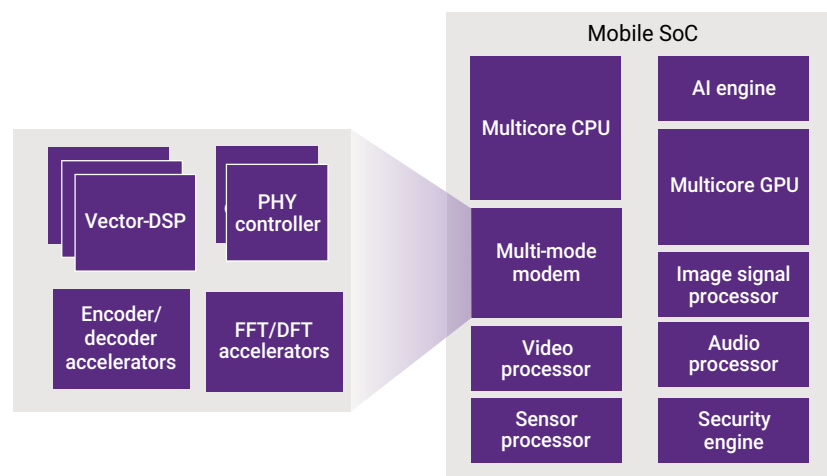


Figure 1: Specialized processors working together in a mobile SoC

This trend has resulted in the increase of more specialized commercial processor IP, which is available from IP providers. One prominent example is the rollout of new processor IP in the domain of embedded vision, optimized for a certain class of vision and machine learning algorithms.

But what if no such specialized processor IP is (yet) available? Traditionally the choice was between selecting an existing processor that was the “closest fit”, or sticking to fixed-function hardware implementations that offer little to no flexibility. However, design teams now are turning to a third option, which is the in-house design of specialized processors and accelerators, tailored to the specific required functionality. Due to their application-specific nature, these designs are known as application-specific instruction-set processors (ASIPs).

The ASIP vs. off-the-shelf processor IP decision is a make vs. buy decision. In general, chip developers choose an ASIP approach in cases where there really is no suitably “close fit” IP for the functionality desired, or where an ASIP can provide a strong competitive differentiation. In addition to the engineering time and effort required to design, optimize, and verify the ASIP design, the work to develop a software development toolchain for programming the resulting design must also be factored in. A tool that automates the process of developing ASIPs, such as Synopsys’ ASIP Designer, addresses both these needs. ASIP Designer not only minimizes the engineering time and effort required to develop specialized processors and associated programming tools, but it also accelerates the path to understanding the performance and efficiency of candidate designs (i.e., design exploration).

So, while the value of an ASIP is well-accepted, it is only with the availability of a tool like ASIP Designer that SoC design teams can make a compelling case to deploy an ASIP instead of a standard processor IP offering or fixed-function RTL.

In this white paper we will describe the ASIP design process, including the needed architectural considerations. And, we will discuss how ASIP Designer overcomes the obstacles that often plague ASIP development, reducing the effort and risk of deploying ASIPs in SoC designs.

Justifying the Development of an ASIP

ASIPs bridge the gap between highly optimized fixed-hardware data path implementations and standard processor IP. As a result, ASIPs can be an ideal third implementation option for design teams.

Depending on the requirements to be met, an ASIP can be developed to execute one specific function (e.g. forward error correction) or it can be used for an entire class of algorithms (e.g. a DSP optimized for wireless algorithms). In the first case, the programmability allows for variants of the individual function (e.g., programming it for LDPC, Viterbi, or Turbo coding). In the latter case, the DSP can be optimized for the specific algorithmic sequences found in the targeted application code (e.g., Layer 1 or Layer 2 baseband processing). In each case, the designer can make tradeoffs to balance performance, flexibility, energy consumption, reusability (or generality), and design time.

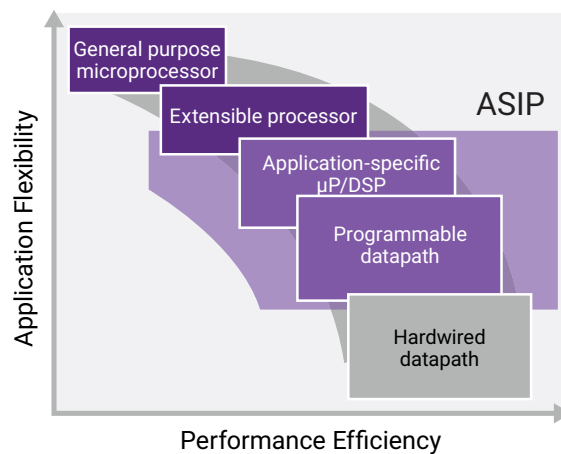


Figure 2: ASIPs deliver greater computational efficiencies than general-purpose processors and more flexibility than fixed-function RTL designs

One fundamental problem in choosing standard processor IP for a very specific function is the overhead involved in mapping the application to the processor’s general-purpose instruction set architecture (ISA). For example, the instruction set may force the application to use an excessive amount of instructions and program memory simply because the function(s) must be implemented using a sequence of general-purpose instructions instead of a shorter sequence of application-specific ones.

In contrast, an ASIP is a combination of a software programmable processor and application-specific functional units, with the architecture optimized for a specific set of functions. An ASIP's ISA is tailored for the efficient implementation of the specific applications it will be running, while still providing enough flexibility to change the operation in software. Because of their architectural specialization combined with instruction-level and data-level parallelism, ASIPs can offer performance and energy characteristics that are superior to general-purpose processors and close to fixed-function custom logic implementation. Unlike fixed hardware implementations, however, ASIPs offer the programmability designers need so they can address evolving SoC requirements.

Architectural Features for an Optimized ASIP

ASIP designers apply the same concepts as in classical hardware design when optimizing their architecture: parallelism and specialization. At the same time, they want to retain full programmability, most often with full C-compiler support. There are two groups of architectural features that can be used to optimize an ASIP design (tailoring to its specific tasks):

1. **Parallelism:** ability to execute multiple functions in parallel. This technique increases performance significantly. Parallelism is a key element in almost all ASIP designs. Parallelism is normally achieved using three different techniques, which can be used individually or in combination (Figure 3):
 - **Instruction-level parallelism** uses an orthogonal instruction set, as in very long instruction word (VLIW) architectures, or an encoded instruction set (which delivers the operational parallelism needed without the overhead associated with VLIW architectures).
 - **Data-level parallelism** implements vector processing (SIMD) which allows operations on multiple data items (e.g. vectors) with a single instruction execution.
 - **Task-level parallelism**, as in multicore/multi-threading implementations, enables the benefits of parallelism when running multiple cooperating algorithms that have different control flows.

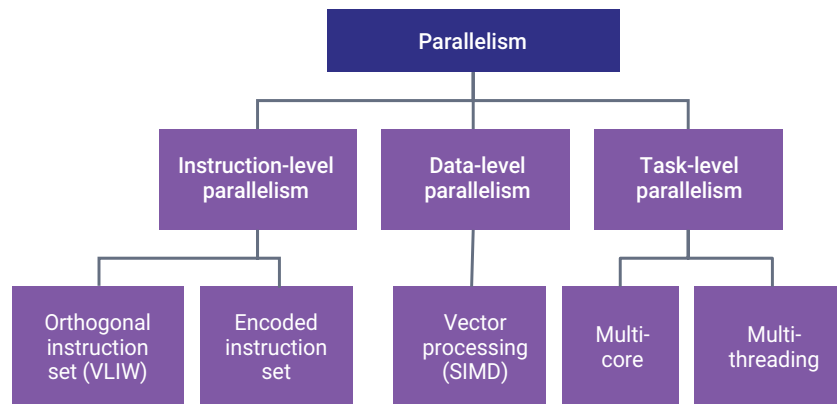


Figure 3: Design Options—Parallelism

2. **Specialization** touches on multiple aspects of a custom processor design, including the instruction set as well as the architecture. It enables designers to perform special functions with one or a few instructions, and customize the pipelines and internal/external memory and register architectures as well as connectivity to those storage locations. Designers can define application-specific data types as well as interfaces customized to the application's data flow and protocol. Figure 4 depicts specialization capabilities.

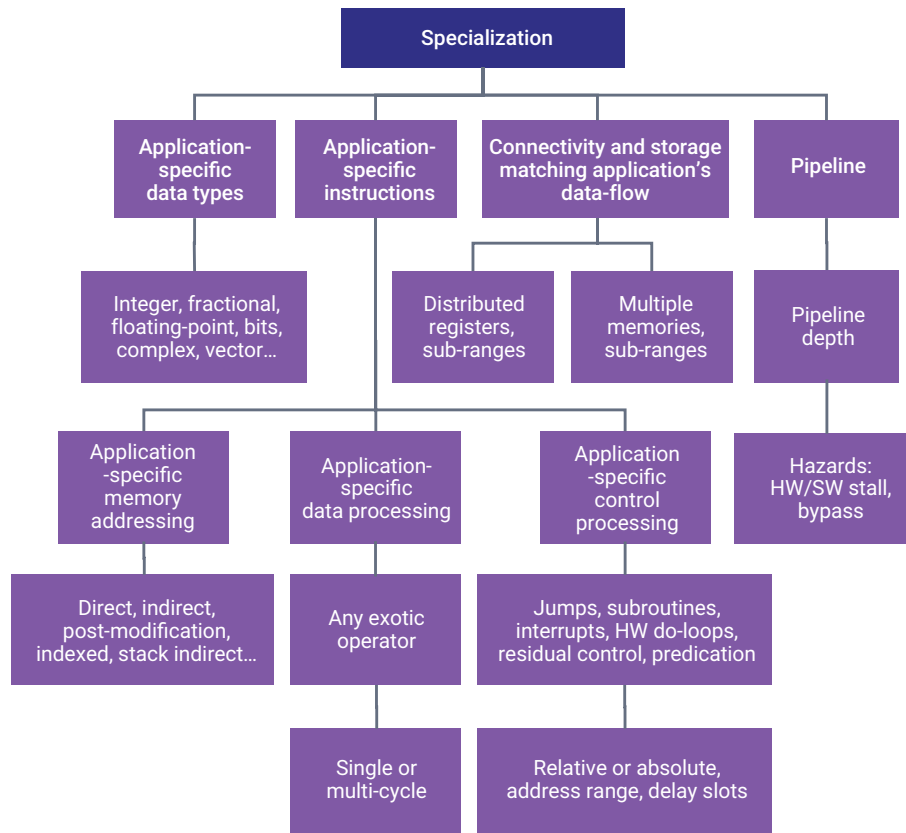


Figure 4: Design Options—Specialization

ASIP Development Requirements

Adopting an ASIP approach comes with the need to first define a processor architecture best suited for the application before spending time and effort on details of the hardware implementation. Therefore, architectural exploration is at the heart of any ASIP design approach. Designers need the ability to rapidly explore the impact of different architectural choices. There are three key requirements for such architectural exploration:

1. Defining the Benchmarks

The benchmarks must be representatives of the application domain targeted, and they need to enable a quantitative comparison of the architectures being considered.

A benchmark must consist of:

- The functional specification, describing the application kernels that need to be implemented. Typically, the benchmarks are represented in C (or another high-level language), to be both easily developed and verified, and to be independent of the architecture. Often, the benchmarks can be culled from existing or open-source code.
- The environment, describing the stimuli needed to exercise the benchmark.
- The performance metrics, such as power, performance, and target frequency at a given technology node.

2. Describing a Candidate Architecture

First, a method must be available to quickly and easily define candidate architectures. Ideally, this is made possible with a model that avoids specification of deep implementation details, so hardware description languages are not suitable.

Second, software tools to map benchmark code onto the candidate architectures are required to systematically evaluate candidate architectures. Moreover, it is impractical to independently develop both the hardware architecture and the software tools from scratch for every candidate architecture, so a method to automate the process is needed. Otherwise, the design team will only be able to explore a limited number of architectures given the constraints of the project schedule.

3. Exploring the Design Space

Design space exploration is a feedback-driven process, where each candidate architecture is evaluated in terms of the appropriate metrics, as defined in the benchmarks. For efficiency, this calls for two key methodologies to be applied:

Compiler-in-the-loop: With the benchmarks described in C, use of a C-compiler capable of compiling the benchmark onto the candidate architecture is necessary. Manually writing the benchmark in assembly language for each candidate architecture is too time consuming and error prone. In addition, a cycle-accurate instruction set simulator (ISS) and a profiler are necessary to execute the benchmarks and analyze the results. The C-compiler, ISS and profiler can be combined with other tools (e.g. debugger, assembler, linker) to form a full software development toolkit (SDK).

For efficient architectural exploration, the SDK should be available early in the design process and be quickly retargetable to the various architectural alternatives that are to be explored. The faster the retargeting, the more architecture options that can be explored and the higher the likelihood of finding the most optimal architecture.

Synthesis-in-the-loop: In addition to using a compiler, ISS, and profiler to analyze the performance of architectures, it is also useful to quickly analyze the hardware cost and characteristics in terms of operating frequency, area, and power efficiency. As with the need to automate the availability of the C compiler, there should be a way to automate the process of generating synthesizable RTL, and use trusted commercial synthesis tools to analyze the characteristics of an actual hardware implementation of each candidate architecture.

To make the transition from standard processors or fixed-function hardware to ASIPs worthwhile, an efficient approach to accelerate the overall architectural exploration and implementation process is necessary. ASIP Designer offers these key features as we will explain in the following section.

ASIP Designer

ASIP Designer is a tool from Synopsys that automates the creation of ASIPs. It offers retargetable compilation and architectural exploration technology as well as the fast simulation and integration with industry-leading implementation flows. Synopsys' ASIP technology has been deployed for hundreds of tapeouts, and in fact, a majority of the top-10 semiconductor companies use ASIP Designer to develop ASIPs for their products.

How the entire development flow of an ASIP device is supported within ASIP Designer, and how it integrates into the Synopsys design and verification flows, is shown in Figure 5.

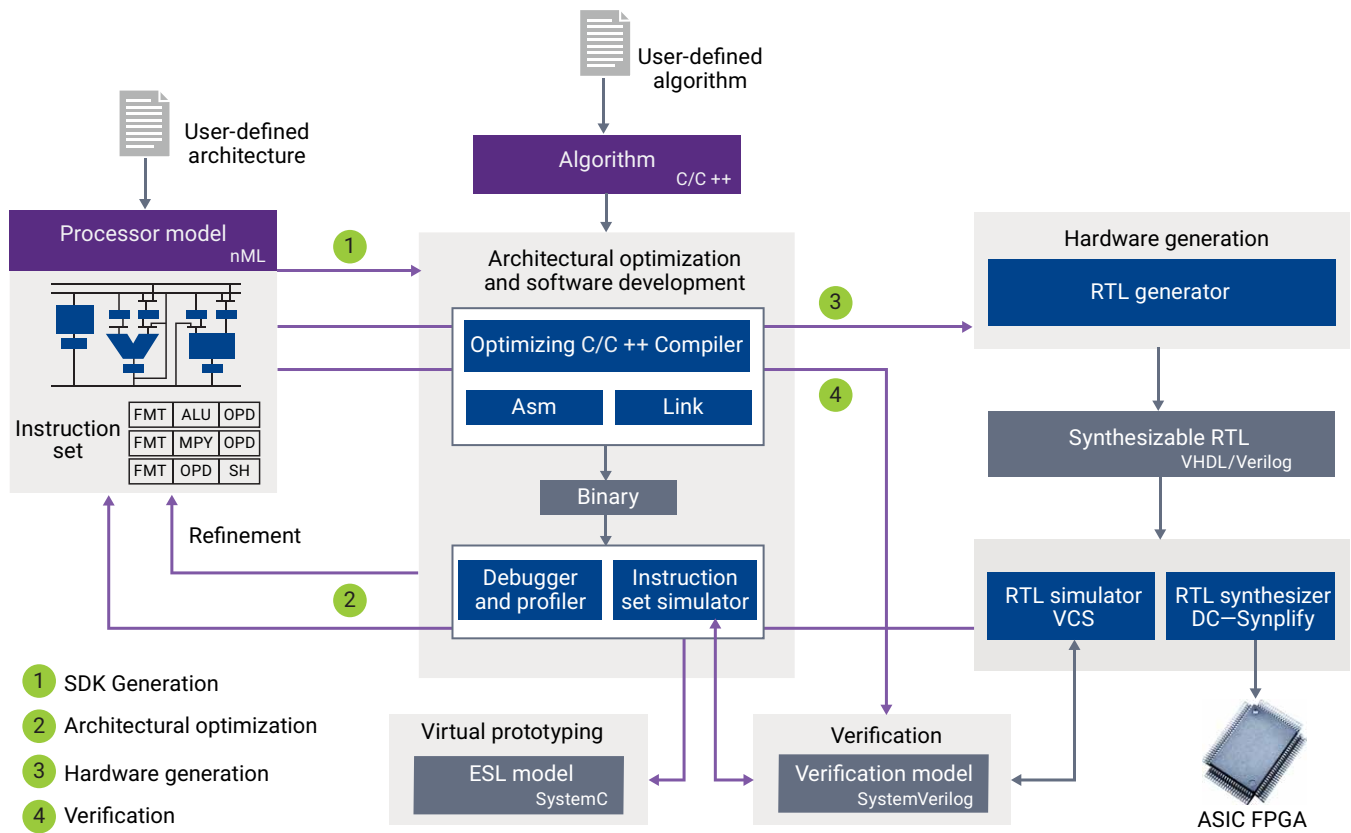


Figure 5: ASIP Designer Tool Flow

Processor Modeling

The ASIP is described using nML, a hierarchical and highly structured architecture description language, that is used to represent ASIP designs at the abstraction level of a programmer's manual. nML might be best compared to VHDL and Verilog, which were defined to describe hardware at the appropriate level of abstraction. In a similar way, nML is defined to efficiently and concisely describe processor architectures. The language is used to model an ASIP architecture in a concise way, defining both the structural characteristics of the design (registers, functional units, signals, etc.) as well as the instruction set architecture. Also, nML allows users to describe the cycle- and bit-accurate behavior of the datapaths and I/O interfaces, providing the designer with full control of the details of the hardware implementation. Developers can define the instruction-set architecture for a wide spectrum of ASIP architectures, ranging from custom microprocessors to highly specialized programmable datapath architectures that can serve as accelerators to a general-purpose microprocessor or DSP.

SDK Generation

In the past, the need to develop and maintain an SDK was the biggest hurdle to overcome when moving to an ASIP due to the number of tools needed. Some of the tools require specific knowledge not generally available in a design team. ASIP Designer significantly simplifies SDK development.

The ASIP's nML description serves as an input to the retargetable SDK (step 1 in Figure 5). The term "retargetable" refers to the fact that the SDK automatically adapts to the processor architecture as defined in the nML description.

The SDK consists of an optimized C/C++ compiler, assembler/disassembler, linker, cycle-accurate as well as instruction-accurate instruction-set simulator and a graphical debugger (suited for instruction-set simulation and on-chip debugging), as shown in Figure 6.

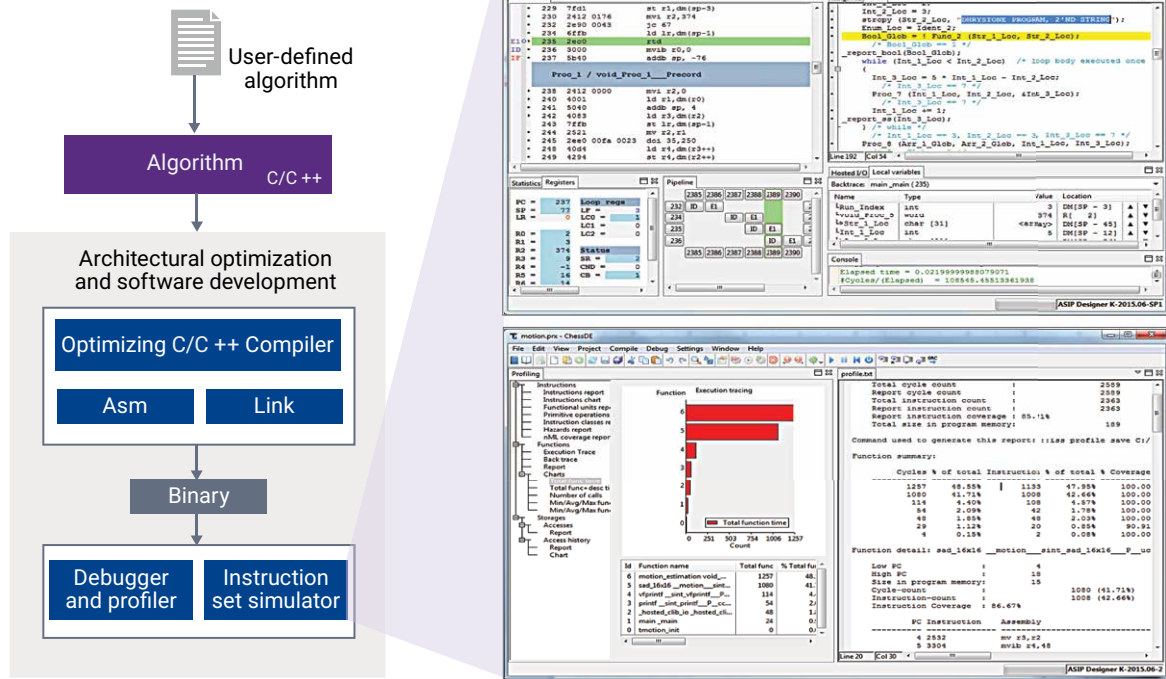


Figure 6: Software-Development Kit (SDK) Components

Automatic retargetability of the compiler is enabled because all compiler optimizations are implemented in a generic way. This is different from compiler frameworks such as GNU or LLVM, where one has to make sure to have architecture-specific compiler backends for each individual processor architecture. The immediate availability of a compiler forms the basis for quick architectural iterations that use “compiler in-the-loop” technology, enabling compilation results to drive architectural decisions in the next iterative step (step 2 in Figure 5).

The development team can debug the software and at the same time provide feedback to the processor designer. The processor can be optimized further because it is now possible to observe its dynamic behavior. It is far more efficient to perform optimization at this level of abstraction before spending effort on a detailed RTL description.

Of course the need for an SDK expands beyond the processor design phase. Once the ASIP design is completed, SoC developers integrating the processor will need to program it. The ASIP Programmer product line makes such SDKs available as a standalone package that is specifically optimized for the ASIP. This gives the end user a packaged, well documented, fully supported SDK.

Hardware Generation and Verification

Once the design meets its functional requirements, ASIP Designer integrates seamlessly with Synopsys implementation and verification tools that take the design from its RTL description to tapeout.

First, developers use ASIP Designer to translate the nML model into fully synthesizable Verilog or VHDL (step 3 in Figure 5). Designers have full control of the hardware, as nML enables a cycle- and bit-accurate description of the processor. At this point, industry-standard design and verification tools can be used, such as Synopsys’ Design Compiler, and VCS. The development team can simulate the design further if required, and then use Design Compiler to generate a gate-level description that is useful to accurately evaluate the circuit’s power requirement and area, and even enter the place-and route process with tools like Synopsys IC Compiler, to explore the risk for routing congestions. This “synthesis-in-the-loop” approach allows for educated decisions, and avoids surprises later in the design process.

Should a problem in the design be discovered during the implementation phase, it is straightforward to go back to the nML description and perform the required hardware and/or software modifications to the model to address the issue, such as power consumption and/or area budgets being exceeded. Because of the single-source entry in nML, both SDK and RTL will always remain in sync.

It's the designer's responsibility to verify the ASIP. There are two aspects to this:

1. Verification of the **processor model** (nML), ensuring that the specified processor model implements the desired behavior
2. Verification of the **RTL model**, ensuring that the generated RTL model implements the processor model correctly

ASIP Designer provides a wide range of assistance for the verification process (step 4 in Figure 5), which includes

- For (1): confirmation of correct test case execution as compared to native execution on the designer's workstation, automatic consistency checks, diagnostic reports analyzing connectivity, hardware conflicts, unused instructions, pipeline hazards, automatic generation of processor specific "one-liner" C programs that check if all units necessary for a compiler are present, and many more
- For (2): automatic generation of *directed random instruction sequences (RIS)* which are directly generated as assembly code, templates of instruction sequences by generating random values for all required fields, automatic generation of coverage points, and many more. Those are provided in SystemVerilog and can be integrated into the overall testbench to be developed by the verification engineer

The effort spent on the verification of the ASIP may strongly depend on the intended use case of the ASIP. The more specific the ASIP, the closer the functional verification resembles that of a fixed-function RTL implementation. The more generic it is, the closer the functional verification resembles the effort a processor IP provider has to spend to ensure proper function for almost any use scenario.

The benefit of the full integration of all the EDA tools required cannot be stressed enough. Porting a version of the design between two tools that are not fully integrated is a major source of errors. Finding the error is time consuming as the development team must often deal with two or more vendors for support.

Conclusion

From self-driving cars to medical devices, from intelligent mobile networks to space applications, from security to virtual reality, virtually every SoC needs or already uses ASIPs. ASIPs address the requirements for specialized processing, where off-the-shelf commercial processor IP cannot meet power-performance-area requirements and fixed-function hardware lacks the needed programmability.

ASIP Designer significantly lowers the barrier to adopt ASIPs for new design projects. Having access to a professional SDK without the need to hire experts on simulators, debuggers or compilers makes a huge difference in design team productivity and time to market. With ASIP Designer, design teams can:

- Replace fixed-function hardware implementations with ASIPs, avoiding the need for designing and verifying complex and inflexible state machines
- Design their own specialized DSPs tailored for specific algorithms, such as image processing, baseband processing, and audio processing
- Create flexible, domain-specific accelerators for high-value and differentiating design blocks such as AI, layer 1 communication, matrix operations

Leading companies in nearly every market have adopted an ASIP approach for SoC design over the last couple of decades. Today, companies worldwide depend on ASIP Designer to assure high-quality ASIP implementations while reducing the effort, expense, and risk of this innovative and highly efficient and powerful design methodology.