Synopsys Insight

SYNOPSYS®
Silicon to Software™

DESIGNING A SMARTER FUTURE FROM SILICON TO SOFTWARE

Subscribe

Issue 2 | 2016

EDA Tools

Verify Software Bring-Up on Your IC Design Before Sending It to the Fab

How to Create a Testbench With Synopsys Verification IP and UVM in 5 Steps

Addressing Physical Design Challenges in the Age of FinFET and Smarter Design

Multi-Level Physical Hierarchy Floorplanning vs. Two-Level Floorplanning

Semiconductor IP

IP for the Era of FinFET and Smart Designs

From Silicon to Software: A Quick Guide to Securing IoT Edge Devices

Customer Spotlight

Movidius Keynotes IC Compiler II Technology Symposium With Vision of Smart, Connected Machines and Networks

Software Integrity

Software Is Everywhere — And So Are the Vulnerabilities

Agile Development for Application Security Managers

Additional Resources

Events

Seminars

Webinars

Synopsys Users Group (SNUG)

Contact us: Insight@synopsys.com

Movidius Keynotes IC Compiler II Technology Symposium With Vision of Smart, Connected Machines and Networks

Mike Santarini, Director of Corporate Communications Strategy, Synopsys

"Is it a Chihuahua's face or a blueberry muffin?" That is one of many problems that can be solved when deep neural networks are implemented close to the image sensors at the network's edge, where SoCs with smart vision sensing can perform low-power image classification—a system envisioned by the high-tech industry now striving to bring higher intelligence to everything connected.

In the keynote at the Synopsys IC Compiler II Technology Symposium (where Intel, Samsung, Broadcom, Mellanox and ecosystem partners ARM and TSMC also presented), Movidius' vice president of marketing, Gary Brown, captivated the audience of 250 IC design experts by laying out Movidius' vision of how a mix of innovative architectural design and advanced IC design tools (such as Synopsys' IC Compiler II™) is enabling companies like Movidius to implement SoCs with greater algorithmic intelligence, advanced vision sensing, and deep learning techniques. These devices bring localized processing and intelligence to the growing number of products connected to (aka at the edge of) the network, thereby offloading an increasing amount of processing from the massive, power-hungry cloud datacenters. This smart processing at the edge in turn enables systems companies to quickly bring to market autonomous cars and drones, augmented reality and virtual reality (AR/VR) products, as well as advanced security systems with intelligent analytics. And it helps companies improve the accuracy, power efficiency, and operating expenditure (OPEX) of deep neural networks (Figure 1).

Having recently announced that it is to be acquired by Intel, Movidius (San Mateo, CA) is a 10-year-old developer of visually-intelligent SoCs. The company's mission is to give the power of sight to machines in this emerging era of smart, connected everything. Movidius' flagship Myriad 2 vision processing unit (VPU) is featured in DJI's Phantom 4 autonomous drone, was selected by Google for machine learning and by FLIR for smart cameras, and is in Lenovo's virtual reality products. It will soon offer a version of its chip on a USB stick, called Fathom, that researchers/developers can plug into USB-supported devices to offload machine learning and execute convolutional neural networks to solve problems in image classification.

DJI's Phantom 4 autonomous drone shows off the advanced processing features of Movidius' Myriad 2. "The Phantom 4 is remarkable," said Brown. "It can take selfies of people in motion (running, biking, or skiing) while navigating around obstacles. Evidently this is a very big and required application today." Users can run the drone in multiple modes, including navigating via GPS and manually. "It is able to detect and move around very well using depth cameras," said Brown. "You can actually attempt to fly it directly into a wall and it will avoid collision on its own. These are pretty advanced intelligent functions for a drone."

While machine intelligence and artificial intelligence algorithms to detect objects and identify people have been around for more than 30 years, design teams previously had to implement those algorithms on large computers. "With all of our pioneering efforts in integrated circuits, we are now able to bring this level of intelligence to devices that are very small, sometimes battery operated, working at the network edge, right next to some image sensors," explained Brown. "It allows companies to create autonomous drones that have to detect things, track things, and classify things."

The technology allows companies to add intelligence to many other applications, such AR/VR systems. “Augmented reality and virtual reality bring together some of the most difficult applications all into one very compact space,” said Brown. “You are wearing a virtual reality product on your head, so it has to detect your gestures and detect your movement and know where your head is positioned precisely to understand where to position things in your field of view.”

Brown points out that eye/gaze tracking in a VR headset is quite complex. “The eye/gaze tracking algorithm is important because it relaxes the workload of the GPU in a system, allowing the GPU to render and display high resolution only where the eye is pointed,” Brown said. “As your eyes are moving, if the display only renders high resolution where you are looking, then it will seem to the user that the whole screen appears to be high resolution. This means you can lower the GPU’s workload and save on power.”

But he added that eye/gaze tracking is just one of the handful of complex applications required to run simultaneously in a VR system. VR brings together a lot of different sophisticated technologies such as depth, tracking, simultaneous localization and mapping (SLAM), gesture recognition, and classification. “All these applications have to be running in parallel, which is why you see a lot of these system tethered to large, high-performance PCs with very high-end GPUs,” said Brown. “Untethering and implementing these applications in a low-power SoC is a tough architectural and design challenge.”

The field of deep learning presents additional challenges and opportunities for smart vision processing SoCs. “Deep learning is a hot topic,” said Brown. “There is a huge need to be able to solve problems that engineers don’t know how to write an algorithm to solve.” Brown said, for example, that most people can tell whether a friend is giving them a fake smile or genuine one, but making that distinction isn’t easy for a machine. “It’s really difficult to write an algorithm to do that, but it is easy for a human being because you have close to 100 billion neurons in your head,” said Brown. “How do you do the same thing in an integrated circuit? What people have done over the last decade is come up with models for these neurons. A neuron is, in a sense, taking a network of say 1,000 neighboring neurons and all those neurons are sending a positive or negative charge. You have to sum over time and sum over space that charge to determine what that output should be. That’s how a real neuron works. But how do you create a model of that? How do you develop a circuit to be able to do the same thing?”

Brown said that the answer is to build a neural network. “We take, for example, a field of pixels. You can cause each pixel to activate and be summed with some weighting by a first level of neurons and then have the outputs go to another level of neurons for additional weighting,” he explained. “You detect in an image, for example, small features with the initial layer of neurons, then move to another layer of neurons and detect bigger and bigger features to ultimately decide: is this field of pixels an A, B, C, D, or E that I’m looking at?”

With a deep neural network approach and extensive testing/training of a network, networks can identify objects and the identities of people in pictures or video, translate languages, and classify audio and can also be applied to many other types of image or vision processing, said Brown.

One of these tests asks the deep neural networks to distinguish between pictures of blueberry muffins and pictures of the faces of tan Chihuahuas (Figure 1). “‘Is it a Chihuahua or a muffin?’—is a difficult problem to solve,” said Brown. “It’s difficult to write an algorithm that detects the difference. [The muffins and Chihuahuas] all seem to have cute eyes and a cute nose.”

“Movidius aspires to create SoCs with the sophistication of the human brain.”

"…today we are seeing an explosion in the use of neural networks because of the sophistication and capabilities in ICs we are designing."

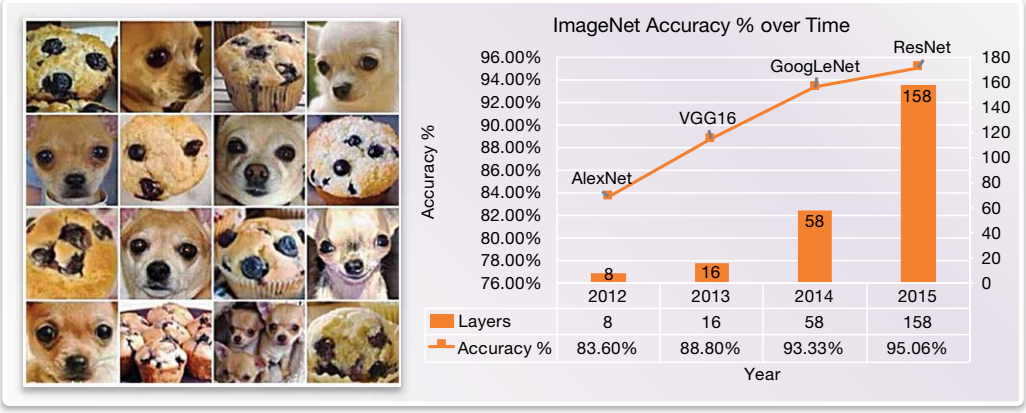


Figure 1. The ability of deep neural networks to accurately distinguish between objects has improved dramatically over the last five years, but further gains in accuracy over the last two has required exponential increases in neural layers and network horsepower

Brown said that deep learning networks have become quite accurate in a relatively short amount of time, but in the last two years, companies have had to add exponentially more neural layers to their networks to achieve marginal gains in accuracy improvements. According to Brown, this is where Movidius SoCs can step in to do essentially localized processing in the many types of vision systems connected to the network.

"What's truly interesting about deep learning today is that the integrated circuits we are designing can now sustain enough performance to be able to run very complex neural networks in real time, where before you were only able to run very simple analog networks to do similar things," said Brown. "But today we are seeing an explosion in the use of neural networks because of the sophistication and capabilities in ICs we are designing."

Brown said that processing at the edge of the network can make a network 1 million times more energy efficient, consume 10,000 times less bandwidth, and have 1,000 times lower latency, while giving networks additional benefits such as improved privacy and fault-tolerance or continuity of service.

Brown said that Movidius aspires to "create SoCs with the sophistication of the human brain."

"If you think about it, doing 10 to the 17th power calculations per second is equal to 100,000 teraflops. If you could do it in a tiny chip, how fascinating would that be? And if you could do it in just 20 watts of the human brain, that's something like 500 gigaflops per watt. That's an amazing amount of performance. We strive to get closer to the performance of the human brain. It so happens that around 50% of the brain's cortex is dedicated to the vision sense," Brown explained.

While not quite as sophisticated as the human brain, the architecture of Movidius' Myriad 2 vision processing unit is highly advanced. In a single device, it combines:

- Configurable imaging and vision hardware engines
- 12 programmable DSPs designed to crunch complex vison and imaging algorithms at very high performance but low power consumption
- Low-power, zero-latency, on-chip memory to support localized processing
- Two RISC CPUs to run the RTOS, firmware, and runtime scheduler

Image processing traditionally takes a camera input through an image sensor pipeline and feeds the image into a programmable processor that does computer vision. "What we did was break the problem into parts and then merge them all together in a more flexible way," said Brown. "We take the image sensor pipeline and implement it with a series of flexible blocks so that you can tap in and out at any part of the pipeline to run any kind of vision processing algorithms you choose. For us, image processing or vision sensing or machine intelligence is all the same thing."

The VPU includes an array of proprietary on-chip vector processing blocks, internal CPUs for scheduling, and a user-programmable CPU for further customization. The VPU also includes several types of interfaces from USB3 to UART to 12 lanes of high-speed MIPI (Figure 2).

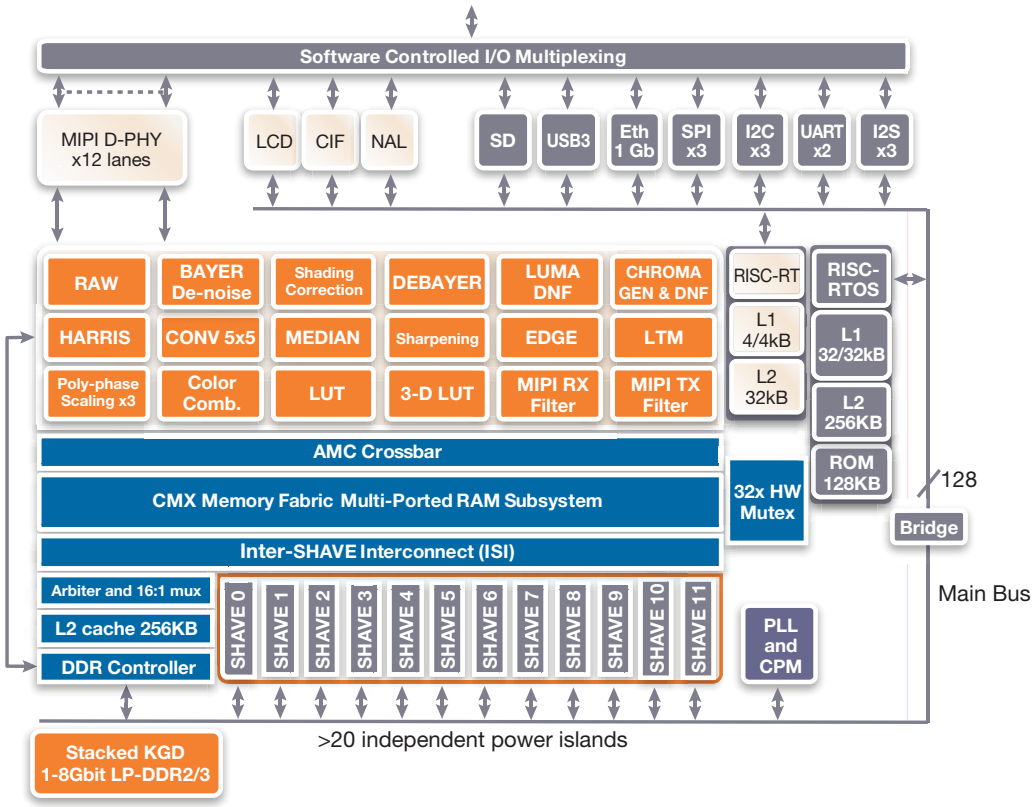


Figure 2. Movidius' Myriad 2 MA2x50 employs an ingenious, sophisticated architecture

Brown said the design, which Movidius implemented on a 28-nm planar process, had more than 20 independent power islands. The company's design team used Synopsys IC Compiler II for physical design of the VPU.

"Synopsys has been a friend of Movidius chip after chip over the years," said Brown. "We have had a number of successful tapeouts together. "Specifically with IC Compiler II, what we have seen is that we can implement a design change in less than a day, where it previously took roughly a week. IC Compiler II gave our team a 10x improvement in turnaround time."

Brown said Movidius' designers also took advantage of the IC Compiler II platform's concurrent clock and data features to achieve faster timing closure while simultaneously lowering the power of clock trees.

The design allowed the company to pack its elaborate architecture into a 6mm x 6mm flip-chip package. "Customers are pretty amazed to see that we packed all of this into a tiny flip chip," concluded Brown. "I would say our success in AR/VR, drones, and surveillance can be partly attributed to the fact that the PCB space required for the chip is minimal. We see a huge opportunity to make smaller yet highly performance-efficient SoCs for machine intelligence applications." ■

About the Author

Mike Santarini, Director of Communications Strategy for Synopsys Corporate Marketing, is a former content strategist at Xilinx and publisher of *Xcell Journal* and *Xcell Software Journal*. Prior to his 8 years at Xilinx, he covered the EDA and semiconductor industries for 14 years at trade publications *EE Times*, *EDN* and *Integrated Systems Design*.

Verify Software Bring-Up on Your IC Design Before Sending it to the Fab

Mike Santarini, Director of Corporate Communications Strategy, Synopsys

The world of IC design verification is becoming even more complex in the era of smart design. As if it wasn't hard enough to verify the functionality of hundreds of blocks, multiple types of on-chip processors and high-speed I/O, hundreds of voltage islands, and thousands of clocks on today's SoC designs, verification teams are finding it's becoming mandatory to verify that an OS will boot up on a design and even run applications before teams can send their design off to the foundry to be manufactured. And leading-edge design teams are already adding more sophisticated testing to this relatively new step of early software bring-up.

"SoC verification is changing dramatically," said Michael Sanie, vice president of marketing in the Synopsys verification group. "Verifying hardware in the context of software is dominating verification activity today." In the past, Sanie explained, verification teams would develop various types of stimuli, run simulation, and use other verification tools to find bugs in their design. They would fix the bugs and then send their design to the fab to make a chip. Once they had silicon back, the software team would bring up software on the silicon and adjust the software until the combined design functioned adequately.

Around 10 years ago that began to change. Companies in the mobile handset market began developing smart phones that required SoC platform ASSPs to support dramatically more software and required SoC design teams to complete their designs in tighter market windows. "They quickly discovered they could no longer wait until the chip came back from the fab to start looking at the software," said Sanie. If the chip came back from the fab and didn't have the functionality or performance to support the software they or their OEM customer planned to run on it, they needed to redesign and respin the chip, which meant they were late to market and took a big hit to profit margins. This, of course, likely means the end of the relationship with the OEM too. Performing software bring-up at this stage also, of course, enabled software developers to get an earlier jump on software development, speeding the handset vendor's overall time-to-market in an increasingly competitive smart handset market.

Sanie added that over the last 5 years, as smart technologies have grown into other application spaces such as embedded vision, automotive, datacenter, SDN/NFV, and IoT, early software bring-up on designs has become the rule rather than the exception. "I don't know of many verification teams today that will send their chip to tapeout without performing some amount of software bring-up on the design before they send the design to the fab — you can't afford not to do it," said Sanie.

Today, how much software design teams run before signing off on the silicon depends on the market they are targeting. Sanie said that because of the accelerating logistics and time-to-market pressure involved in most markets, most companies don't run their entire software stack on the design before sending the design out to be manufactured. "At the very least, they do check that they can boot up Android or Linux on the design before the design is sent to the fab," he said.

In the age of smarter technologies, in which more sophisticated hardware and growing stacks of software are combined to perform more complex tasks and a wider and greater number of simultaneous tasks, bringing up software and confirming it runs seamlessly with the targeted hardware seems to be evolving rapidly from a "nice to have" to a mandate in some industries.

The automotive industry's ISO 26262 standard for automotive design is a prime example. It recommends that automotive suppliers perform extensive IC-level HW/SW safety testing and recommends that teams document their testing methods and findings in detail (see Insight 1, 2016 automotive special issue). And in a growing number of industries (IoT, smart cities, smart

infrastructure, autonomous driving, etc.) that employ smarter, connected systems, security testing is becoming mandatory, not just safety testing. While the standards do not explicitly mandate pre-silicon HW/SW verification today, the increasing complexity of the devices and software running on them seem to indicate they will become mandated in the not-too-distant future.

Arturo Salz, a scientist in the Synopsys Verification Group, points out that in many smart systems, the close coupling of complex ICs and complex software stacks are becoming more important. Today's smart SoCs can have hundreds of voltage islands, thousands of clocks, multiple processors, and run in multiple modes. What's more, to serve more markets with one piece of silicon, semiconductor companies are increasingly creating heterogeneous processing platforms, essentially everything-including-the-kitchen-sink ASSP SoCs, that can run many different types of OSs and RTOS and software. A software glitch or a malicious program can cause a processor core to excessively overclock and burn itself up or quickly wear out a system's battery, so silicon has to have wider tolerances and be able to return to a safe and recoverable state if something in software goes haywire.

Sanie added that while software bring-up is becoming mainstream and even mandatory in some markets, it is still "early days" for what tasks design teams perform during this relatively new step of early software bring-up. "We are starting to see customers increasingly perform power profiling and power estimation during this phase," said Sanie. "Smart phone manufactures will emulate cell phone operations for days, running all different kinds of apps to identify power consumption characteristics and proactively identify potential issues. From these runs, they can get a better view of the power profile of their overall system and decide if they need to adjust the voltage islands on the chip design or make a system-level change like go to a bigger battery, etc." Salz added that software bring-up and power profiling require an exceptional level of hardware verification and software testing. "They can be tested independently but must also be tested together," said Salz.

But to bring up ever-more stacks of software on the designs of today's massive and massively complex IC, SoC, and heterogeneous platform ASSP designs requires an enormous amount of verification compute horsepower as well as a methodology adjustment to verify designs are functionally correct. "To bring up Android requires around 40+ billion or more cycles," explained Sanie. "That's why teams are moving to hardware for the early software bring-up." Sanie said it's also a big reason why emulation and FPGA-based prototyping are becoming mainstream and the market is growing.

Indeed, in their EDA earnings report for calendar Q2 2016, the big three companies in the EDA industry all cited growth in the emulation sector, indicating the need for hardware-assisted verification is growing. Sanie estimates that some of this growth can be attributed to design teams simply having more gates/transistors to work with (per Moore's Law) and a need for verification to handle these higher capacity designs. He predicts early software bring-up will be a new growth driver for the verification space, as the coupling of complex hardware and larger software stacks drive competition in the race for companies across most markets to bring smarter systems to market.

Sanie said that he believes Synopsys' ZeBu® is the fastest growing emulation family in the industry and asserts that ZeBu emulation is also the fastest in terms of raw performance, with design clocks running up to 5MHz. That's well above the 1 MHz performance design teams have traditionally been able to get out of emulation systems. Meanwhile, the HAPS® system can run a system clock in the range of 15MHz to 20MHz, typical. Sanie said that increasingly teams use the emulation systems for bring-up because, it gives teams a simulation-like environment to verify their designs. Design groups will use HAPS prototyping for developing software in the context of the entire system.

Sanie continued that the growth in Synopsys' emulation business is not based solely on the raw performance of ZeBu and HAPS systems but also on the completeness and seamlessness of Synopsys' integrated verification platform — the Verification Continuum™ (Figure 1).

"I don't know of many verification teams today that will send their chip to tapeout without performing some amount of software bring-up on the design before they send the design to the fab—you can't afford not to do it."

"Sanie said that he believes Synopsys' ZeBu is the fastest growing emulation family in the industry and asserts that ZeBu is also the fastest in terms of raw performance, with design clocks running up to 5MHz."

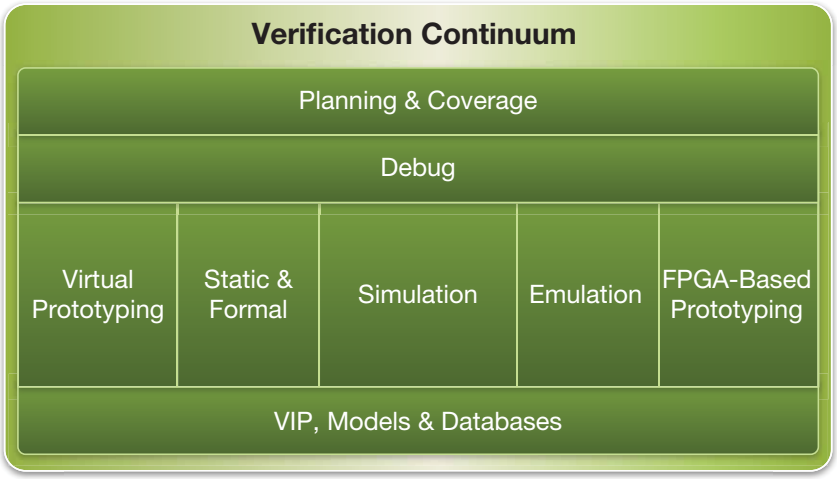


Figure 1. Synopsys' Verification Continuum platform offers new levels of integration and time-to-market savings.

"The Verification Continuum comprises best-in-class point tools at every step in the verification flow," said Sanie. "Best-in-class point tools throughout the verification portfolio is in and of itself compelling, but over the years, we've done a lot of work to integrate the tools into a seamless platform so that the verification tools across the platform not only have the fastest engines, but also share several native integrations including unified compile and unified debug capabilities. We architected these native integrations to significantly reduce the time it takes design teams to move a design from simulation to emulation to FPGA prototyping and to tackle debug challenges throughout the flow."

Sanie explained that with the unified compile capability, verification engineers can take, for example, the same RTL design block they implement in the VCS® simulator and move that block over to the ZeBu emulation system or to the HAPS FPGA-based prototyping system, and the design will maintain the same functional integrity across all three verification models (but, of course, run the functions faster in hardware emulation and prototyping). "With the unified compile capability, whatever you have in the simulation — whether it be RTL file, your UPF file, or whatever you have — you can use that same exact design in emulation and FPGA prototyping, no longer needing to modify the design itself to be able to bring it up in emulation or FPGA prototyping," added Sanie. Salz further added that simulation solutions typically work sequentially and are event driven (per the Verilog semantics), while hardware is highly parallelized. Unified compile includes the engines that allow the simulation, emulation, and FPGA prototyping to work congruently (Figure 2).

What's more, Sanie said that unified debug with Verdi® debug (the industry's de facto solution) means that verification teams can perform IC debug with same quality and integrity in whichever verification platform they choose. ■

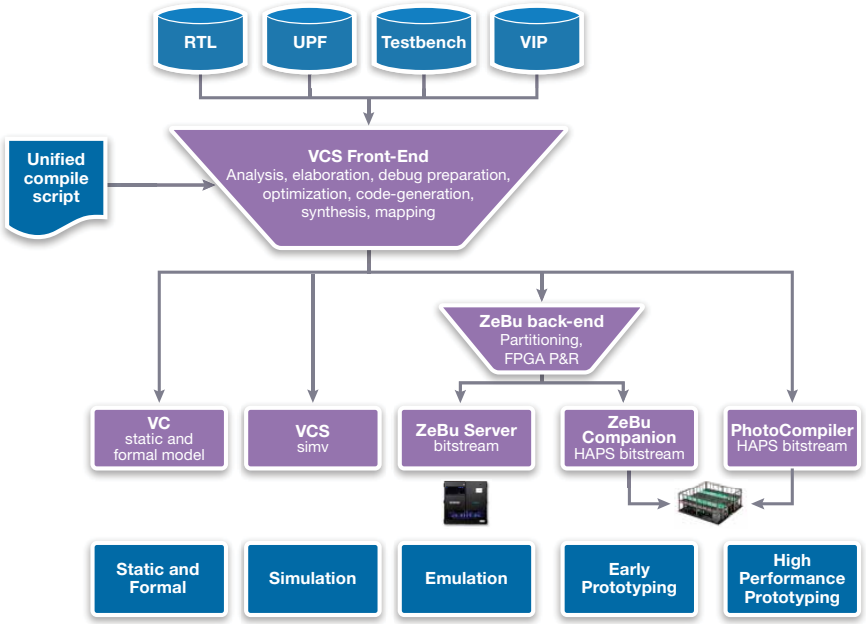


Figure 2. The Verification Continuum platform's unified compile capabilities allows verification teams to move designs from simulation, emulation to FPGA prototyping with ease.

Tom Borgstrom, senior director of emulation product marketing at Synopsys, is quick to point out that no two customers do verification the exact same way and in many cases no two verification teams within the same company perform verification the same way. "The Verification Continuum platform doesn't dictate how teams must run verification," Borgstrom said. "It gives verification teams a lot of options to run whatever methodology they employ, but will also help them run their methodology more effectively with the highest quality tools and reliable flow between those tools."

For example, some companies may have multiple, very large designs and very large verification teams. They may start projects by having verification team members looking over the shoulders of designers and performing block-level simulation with VCS and Verdi tools immediately after designers complete each minor block. They may have a team of associates whose job it is to connect those blocks/builds into larger blocks and then run the larger blocks through simulation and emulation and Verdi debug, repeating the build and verification process until they have a complete chip design. They then run the entire chip design on an emulation system and debug the design for hardware bugs. Once they have the hardware verified and stable, they then have the software team bring up the required software before signing off on the chip design. Once the software team confirms the software bring-up works to their satisfaction, the design team sends the chip to the fab and the software team continues to the emulation system or FPGA prototype to develop and bring up more layers of software at a chip and full system level. "Some companies even use emulation systems to help diagnose post-silicon failures," said Borgstrom.

Other teams may want to perform software bring-up even faster and employ early software bring-up methodologies, in which verification teams typically run functional or behavioral models of known-good reused blocks in simulation and run new RTL blocks they've built for their new chip design in emulation. In doing so, they can typically speed up the overall clock cycle performance of the design to enable software teams to more quickly check if the chip design will be able to boot up their targeted OS or even bring up application software the design must run.

While silicon is being manufactured, they can also create a C-level model/virtual prototype of their design to allow both in-house software developers and end-customers get an early jump on application software development. ■

How to Create a Testbench with Synopsys Verification IP and UVM in 5 steps

Amit Sharma, Director, Corporate Applications

Verification is one of the biggest challenges for IC designs and traditional methods have run out of steam. Writing individual tests is impractical for today's large, complex designs because the state space and number of test conditions are simply too large for verification engineers to code by hand, leading to insufficient test coverage. Limited project resources and time-to-market pressure is compounding the problem. The repercussions are clear — first-pass silicon success is harder to achieve.

To address these challenges, IC verification teams are turning to advanced and unified verification methodologies that leverage multiple technologies. Constrained-random verification leverages compute resources and functional coverage technology to provide more testing with less test code development. Setting up a constrained-random test environment, however, can seem like a difficult task, especially when you consider that environments need to be flexible, scalable, and reusable. The infrastructure for constrained-random verification requires more planning and structure, but the benefits in the end are well worth the investment.

Synopsys Verification IP (VIP) leveraging the Universal Verification Methodology (UVM) provides verification engineers with the tools to rapidly build a well-architected, advanced verification infrastructure. Let us start by giving you an overview of the benefits of Synopsys VIP and a brief introduction to UVM. This will supply the background for the discussion of the five simple steps to creating a complete testbench environment for effective constrained-random verification. The concepts and techniques we will explain and demonstrate, and the code examples we will provide, will serve as templates for real application techniques you can use in your designs.

For this article, we will assume you have working knowledge of SystemVerilog and object-oriented programming.

Benefits of Using Synopsys VIP With UVM

The Synopsys VIP models alone offer many benefits, as they provide proven protocol functionality for verification engineers. When used in a UVM flow, verification teams can attain a number of benefits.

The first benefit is that UVM promotes a modular testbench architecture and provides a standard object-based interface that connects components within a test environment. Better modularity simplifies development and reduces maintenance. Synopsys VIP models provide both protocol functionality and control features in a complete, self-contained package that fully supports the modular architecture of UVM and simplifies the development of UVM testbenches. This gives verification engineers a modular foundation layer over which they can quickly build a robust testbench.

Efficiency of abstraction is another benefit. UVM is based on object-oriented programming (OOP). Synopsys VIP abstracts protocol transactions into objects and provides an object-based interface, allowing engineers to work in logical protocol terms without worrying about implementation details. With protocols becoming more complex, this abstraction is a big boost, as dealing with the details of a standard protocol is time-consuming and does not add value to the end product. Another benefit of the modular, layered approach is that it allows engineers to stack components to create complex systems. For example, a typical webcam transports video data stacked on top of the USB protocol.

Rapid creation of complex tests is the third benefit. While modularity enables the construction of complex test infrastructures, constrained-random verification and efficiency of abstraction allow the easy development of complex tests. Engineers can easily create tests that exercise different scenarios within a given set of constraints. In encapsulating protocol functionality, Synopsys VIP allows engineers to code with abstracted objects where they can easily create tests for intricate and complex combinations of transactions. Engineers can use these sequences to mirror real-world traffic, create stress or corner-case conditions, or simply cover a wide range of conditions. They can create more conditions by simply letting the test run for a longer time.

Increased reuse is another benefit of using Synopsys VIP with UVM. Synopsys VIP models are inherently reusable blocks that all have the same look and feel, simplifying the integration of multiple components. Meanwhile, UVM is architected for maximum reuse, and it fosters testbench code that maximizes reusable components. UVM even provides a template for a standard testbench flow, which engineers can customize to suit their specific needs. The UVM base classes provide reuse through inheritance.

Introduction to UVM for SystemVerilog

Now that we have described the key benefits of using Synopsys VIP models with UVM, it is time to dive into a more detailed look. We will look at a few basic constructs to gain a conceptual understanding of the basics and get you going on your way to creating more effective testbenches. We'll then dive further into the nuts and bolts of building a first testbench using Synopsys VIP and UVM. We'll show how to apply the methodology and concepts using actual code.

So what is UVM? UVM stands for Universal Verification Methodology. It consists of three things: a set of base classes, a verification methodology, and a modeling approach guideline.

The guiding principles of UVM are to emphasize constrained-random verification, maximize reuse, minimize test-specific code, and enable more testing with less code.

Verification Methodology

UVM provides a template for an advanced verification methodology. Supporting constrained-random verification is different than using a directed, sequential flow. UVM provides a blueprint methodology so that testbench code is effectively organized for constrained-random verification. And the resulting structure also supports directed testing, so it serves all verification needs.

The shift to object-oriented programming techniques, the dynamic nature of constrained-random verification, and the need to develop code efficiently are all encapsulated in the UVM. To achieve its goals, UVM prescribes an overall testbench organization, which impacts the way that testbench code is written. Here are some highlights:

- Most of the code is dedicated to setting up dynamic processes in advance. Everything is already programmed by the time the test is started.
- Test conditions are controlled by constraints instead of procedural code.
- Tests react to significant events dynamically. For example, a testbench must be able to detect the end of the testing sequence since the length of the sequence is not pre-defined. Checking mechanisms are dynamic. Scoreboards or other self-checking mechanisms store information on-the-fly and sometimes perform all checks on-the-fly.
- Objects use the UVM base classes whenever possible to maximize reuse and guarantee interoperability.
- A standard testbench sequence template is used (build-configure-execute-report). Each testbench uses the template but fills in the details according to its needs.

- Programming orientation shifts from procedures to objects. We won't discuss this in detail, because it is inherent in the adoption of SystemVerilog and is outside the scope of this article.

UVM Verification IP

In an object-oriented programming environment, a set of base classes form the foundation for the entire system. Base classes provide common functionality and structure. Because most objects will be derived from them, a well-architected set of base classes is essential to the end goal of an effective programming environment.

The classes provided by UVM are specifically designed for verification. They provide the base functionality that verification engineers need for simulation (such as logging), and they support any sort of verification task. The Synopsys VIP models are based on the same UVM classes that are also available for the end user, allowing easy integration of user code and Synopsys VIP in a verification environment. Further, UVM provides an actual implementation of its base classes and is not simply a set of guidelines or recommendations.

A simulation VIP written in UVM presents a configuration and transaction-level interface that is used by testbench environments and test cases to interact or monitor a design under test (DUT), usually through a signal-level interface. Using a simulation VIP to implement a simulation environment allows engineers to write test cases at a much higher level of abstraction.

Agent

In UVM, a VIP may be called a UVM verification component (UVC) or simply an agent. UVM VIPs are implemented as agents. UVM agents have a well-defined architecture: they encapsulate a sequencer, a driver, and a monitor. In active mode, all components encapsulated by the agent are instantiated and are running. In passive mode, the sequencer and driver are not instantiated and only the monitor is running. We use the uvm_agent base class to create an agent.

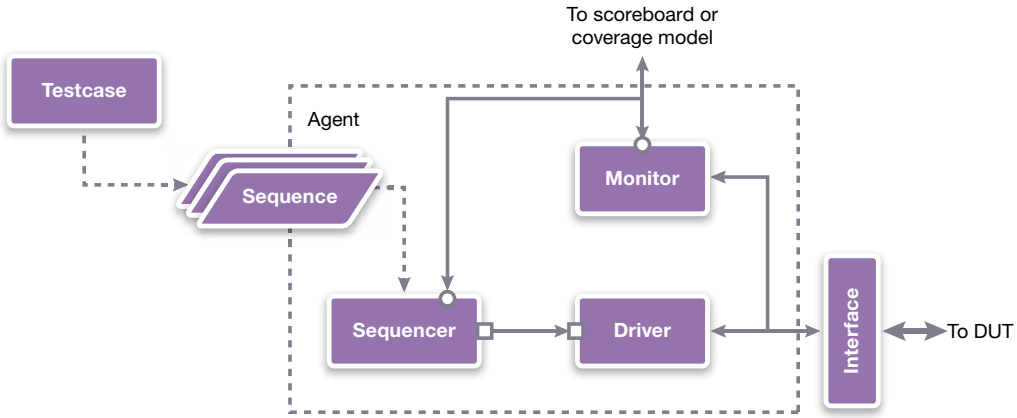


Figure 1. UVM agent architecture

Configuration: An agent is usually configured through a configuration object, passed via the UVM configuration database. The agent configuration object is usually obtained from the environment configuration object.

Sequencer: The sequencer is a component that executes one or more sequences concurrently. A sequence, when executed, creates sequence items (transactions) that the sequencer sends to the driver for execution. The sequencer arbitrates between the different sequences that are being executed in parallel, selecting one to generate its next sequence item (transaction). Once the sequence item is generated, it is then sent to the driver for execution. The sequencer is also responsible for collecting the transaction responses and forwarding them to the appropriate sequence.

Driver: The driver executes transactions (sequence items) as they are submitted by the sequencer according to the semantics of the protocol it implements and the transaction description received from the sequencer. A typical driver drives the signals encapsulated in the interface according to the protocol specification. The actual values driven on data signals, and the relative timing of control signals, will be taken from the corresponding properties in the transaction description. If the protocol allows it, a driver may execute more than one transaction at the same time or in a different order (should the sequencer supply multiple transactions before the previous ones complete their execution).

Monitor: The monitor observes the signals encapsulated in the interface and interprets their transitions and values according to the protocol semantics. It reconstructs transaction descriptors for every observed transaction and publishes them on its analysis port. The published transactions can then be used by a scoreboard to determine if the overall operation of the DUT is functionally correct, or by a coverage model to record which conditions have been applied to the DUT. The monitor also performs protocol-level checks and reports any observed functional or timing violations using warning, error, or fatal messages.

The execution of every test case is divided into several orchestrated or sequential phases. Typical phases include resetting the DUT, configuring the DUT, executing the actual test, and cleanly shutting the DUT. UVM has a well-defined phasing mechanism that can, when properly used, meet the demands of the most complicated test cases. The following figure shows all the different phases through which the UVM structural components are sequenced.

"UVM provides a blueprint methodology so that testbench code is effectively organized for constrained-random verification."

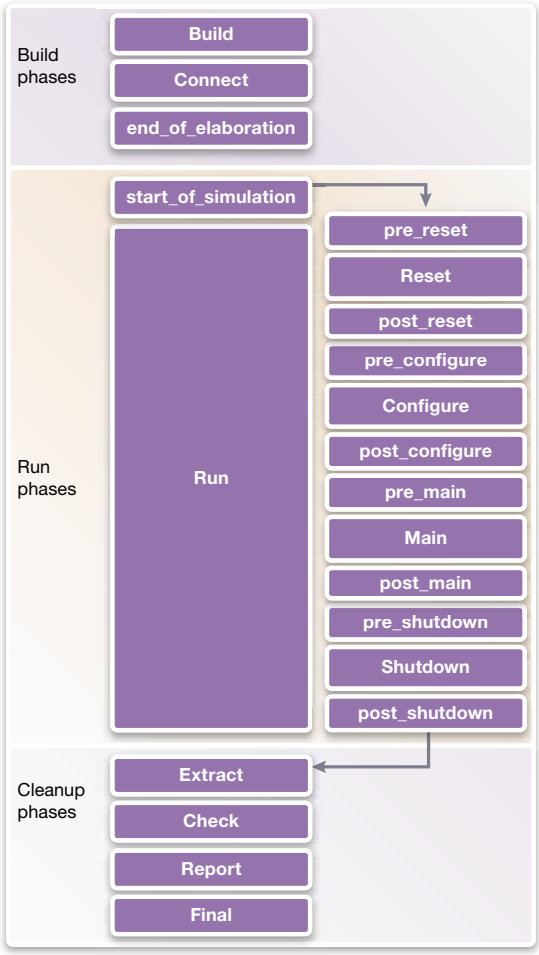


Figure 2. UVM phases

For a Synopsys VIP user, the modeling approach defined by UVM is already incorporated into the models. This means that the models are UVM compliant and will operate seamlessly within a UVM environment.

Five Vital Steps to Using Synopsys VIP in UVM

Now let's apply a few basic UVM concepts and techniques to quickly achieve a basic constrained-random testbench. The first five steps to using Synopsys VIP and UVM are:

1. Create/instantiate a test environment.
2. Connect the VIP to the DUT.
3. Instantiate and configure the VIP.
4. Create a test sequence.
5. Create a test.

To illustrate these methods in practical use, we'll draw code snippets from a complete example testbench. The testbench shows typical Synopsys VIP and UVM usage, and it highlights the concepts and techniques described. Although the example code uses Synopsys VIP for the ARM® AMBA® AXI3™ protocol specification, the methods we show here are not specific to the protocol. You can use them with any of the Synopsys VIP models. To obtain the example code, please refer to the instructions at the end of this article.

Step 1: Create/Instantiate a Testbench Environment

An environment encapsulates all of the agents, monitors, checkers, and scoreboards necessary to stimulate, observe, and check the correctness of the design under verification. Environments are strictly structural components. They instantiate and connect the various components needed to verify a design, but they do not contain any actual behavior toward accomplishing that verification. An environment is implemented as a class, extended from the uvm_env base class. The environment class has public members for each of the individual agents, monitors, checkers, or scoreboards it encapsulates. These components are instantiated in the build_phase() method and are appropriately interconnected in the connect_phase() method. An environment class should not implement any run-time phase.

For the Synopsys VIP, this environment encapsulates all of the VIP components and implicitly constructs the required number of protocol agents as specified by its system configuration object. As we will see, the vast majority of code in a UVM testbench is contained in the environment where it can be reused by other tests or projects.

The user inherits structure and base functionality from uvm_env and yet can still customize where needed. Furthermore, the customization is under the user's control. This is a common theme throughout UVM and the Synopsys VIP models.

```
class test_env extends uvm_env;
   svt_axi_system_env axi_system_env;
   ...
   scoreboard           sb;

   function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      ...   axi_system_env =
      svt_axi_system_env::type_id::create("axi_system_env", this);      ...
      sb = scoreboard sb::type_id::create("sb", this);
      ...
   endfunction

   function void connect_phase(uvm_phase phase);
      super.connect_phase(phase);

      axi_system_env.master[0].monitor.item_observed_port.connect(sb.axi_
master0_aip);
      ...
   endfunction
   ...
endclass
```

The following are steps to instantiate the AXI™ system environment in your testbench environment.

```
svt_axi_system_env axi_system_env;
axi_system_env = svt_axi_system_env::type_id::create("axi_system_env", this);
```

In the build_phase() method, obtain the top-level configuration object (see Step 3) from the configuration database. Instantiate each sub-component (agents, sub-environments, scoreboard), setting the configuration information for each sub-component appropriately. To improve the ability to debug, use the data member handle name as the component instance name when calling create().

"The shift to object-oriented programming techniques, the dynamic nature of constrained-random verification, and the need to develop code efficiently are all encapsulated in the UVM."

```
function void build_phase(uvm_phase phase);
      super.build_phase(phase);

      if (!uvm_config_db#(tb_cfg)::get(this, "", "cfg", cfg)) `uvm_fatal(...)
      master0 = svt_axi_master_agent::type_id::create("master0", this);
      uvm_config_db#(svt_axi_master_configuration)::set(this, "master0", "cfg",
cfg.axi_cfg.master_cfg[0]);
      ...
      sb = scoreboard sb::type_id::create("sb", this);
      uvm_config_db#(tb_cfg)::set(this, "sb", "cfg", cfg);
   endfunction
```

In the connect_phase() method, connect the analysis ports of the relevant agents to the analysis exports of the scoreboard.

```
   function void connect_phase(uvm_phase phase);
      super.connect_phase(phase);

      axi_system_env.master[0].monitor.item_observed_port.connect(sb.axi_
master0_aip);
      ...
   endfunction
```

To be reusable in eventual higher-level environments, the top-level environment class should only implement structural and connectivity aspects of the testbench.

Step 2: Connecting the VIP to the DUT

The DUT harness instantiates the DUT and provides all of the necessary support to drive and monitor all of its I/O signals. It is responsible for generating clock and reset signals, and for connecting the I/O side of the verification IP to the DUT.

The following are the steps to establish a connection between the VIP to the DUT in your top-level testbench:

- Include the standard UVM and VIP files and packages

```
`include "uvm_pkg.sv"
`include "svt_axi.uvm.pkg"
`include "svt_axi_if.svi" //top-level AXI interface
import uvm_pkg::*;
import svt_uvm_pkg::*;
import svt_axi_uvm_pkg::*;
```

- Instantiate and connect the top-level AXI interface to a system clock

```
svt_axi_if axi_if();
assign axi_if.common_aclk = <system clock>;
```

- Connect the top-level AXI interface to the DUT and the AXI system environment

```
dut dut_inst(axi_if); uvm_config_db#(svt_axi_vif)::set(uvm_root::get(), "uvm_test_top.
env.axi_system_env", "vif", axi_if);
```

The uvm_config_db command connects the top-level AXI interface to the virtual interface of the AXI system environment. The "uvm_test_top" represents the top-level module in the UVM environment. The "env" is an instance of your testbench environment. The "axi_system_env" is an instance of the AXI system environment (svt_axi_system_env).

- Connect the reset pin of the AXI master and slave agent interfaces to the desired reset signals

```
assign axi_if.master_if[0].aresetn = <User-select Reset Signal>;
assign axi_if.slave_if[0].aresetn = <User-select Reset Signal>;
```

In the sections that follow, the individual steps in the test sequence are shown in detail.

Step 3: Instantiating and Configuring the VIP

Synopsys configures Synopsys VIP models using configuration objects that are provided and ready for use. The environment configuration classes are extended from the uvm_object class. All public members are declared as 'rand' in that class. Additional, sub-environment configurations may also be members of this class. The sub-configurations would be created within the constructor of the top-level configuration object. Engineers can handle configuration objects just like any other objects, so the objects can be randomized and passed as an argument to a method. The objects come with constraints so they adhere to protocol limits. These can be controlled or extended as desired to create specific test conditions, or used as is to produce a wide range of stimulus.

We can customize an AXI system configuration class by extending the AXI system configuration class (svt_axi_ system_configuration) and specifying the required configuration parameters.

For example:

```
class cust_svt_axi_system_configuration extends svt_axi_system_configuration;
 function new (string name = "cust_svt_axi_system_configuration");
 super.new(name);
 // Create a single AXI master agent and a single slave agent
 this.num_masters = 1;
 this.num_slaves = 1;
 // Create port configurations
 this.create_sub_cfgs(1,1);
 this.master_cfg[0].data_width = 256;
 this.slave_cfg[0].data_width = 256;
 this.master_cfg[0].id_width = 4;
 this.slave_cfg[0].id_width = 4;
 endfunction
endclass
```

Once the customized configuration object is created, the instance needs to be passed to the system environment (instance of svt_axi_system_env) in the build phase of your testbench environment.

```
cfg = cust_svt_axi_system_configuration::type_id::create("cfg");
uvm_config_db#(svt_axi_system_configuration)::set(this,
"axi_system_env", "cfg", cfg);
```

Once set, in the build_phase() method of the testbench environment (see Step 1), obtain the top-level configuration object from the configuration database. You can then use the configuration object values to instantiate each sub-component (agents, sub-environments, scoreboard, etc.).

Step 4: Creating a Test Sequence

The object-based interface provided by Synopsys VIP and UVM creates a subtle yet important shift in how constrained-random generation can be used in the context of creating the necessary traffic patterns for the various standard interfaces. Building on the fact that the models abstract protocols into transaction objects, specifying test conditions is a matter of generating constrained-random objects. The shift here is that defining the transactions occurs in protocol terms and uses native language syntax (constraints and assignments), so verification engineers can think in protocol terms, not model details. This abstraction allows a more natural and efficient thought process.

To enable this, the VIP provides a base sequence class for the different protocol agents. For example, the AXI master agent would have the svt_axi_master_base_sequence and the AXI slave agent the svt_axi_slave_base_ sequence. You can extend these base sequences to create test sequences for the AXI master and slave agents. The VIP also provides a sequence collection, which essentially is a list of random and semi-directed sequences that extends from these base sequences. These sequences, when executed, create sequence items (transactions) that the sequencer sends to the driver for execution. The sequencer arbitrates between the different sequences that are being executed in parallel, selecting one to generate its next sequence item (transaction). Once it generates a sequence item, it is then sent to the driver for execution.

The sequence item created within a sequence running on a sequencer is a blueprint for randomization and the template for the generated objects. When using Synopsys VIP models, you can manipulate them by extending the provided base transaction (extended from uvm_sequence_item) objects and applying your own user-defined constraints. Using the 'factory override' mechanism in UVM ensures that when the transaction is randomized, the user constraints will be used and, through inheritance rules, that the extended objects can be driven onto the driver by the sequencer. Synopsys VIP models also support the addition of user data members in these transaction objects, allowing user customization of the transactions themselves. The details of this are outside the scope of this article.

Step 5: Creating a Test

The Synopsys VIP models all share the same features regarding test control. You can create a VIP test by extending the uvm_test class. In the build phase of the extended class, you construct the testbench environment and set the sequences to run on the respective sequencers and in the relevant phase. In the context of the AXI VIP, you can create and run a simple test as follows:

```
class random_wr_rd_test extends uvm_test;
 // build_phase
 env = axi_basic_env::type_id::create("env", this);
       uvm_config_db#(uvm_object_wrapper)::set(this,  "env.axi_system_env.
master*.sequencer.main_phase", "default_sequence", axi_master_wr_rd_
sequence::type_id::get());
uvm_config_db#(uvm_object_wrapper)::set(this, "env.axi_system_env.slave*.
sequencer.run_phase",
"default_sequence", axi_slave_mem_response_sequence::type_id::get());
        //
       endclass
```

In the above code snippet, we have used a sequence, axi_master_wr_rd_sequence_, which is part of the AXI VIP sequence library to run on the 'main_phase' of the sequencers of all the master agents configured and instantiated in the AXI system environment. Similarly, the axi_slave_mem_response_sequence, which is also part of the sequence collection, is configured to run on all the active slaves. Note, you must set a slave response sequence for active slaves in the run phase.

Notice how small the test-specific code is when you use UVM! Most code is in the environment, which is a reusable component. The test-specific code is minimized and as much code as possible is common so it is not replicated unnecessarily. This yields a smaller code base to maintain.

The code shown in this article is from an example testbench, which ships with the AXI3 VIP installation, and is a complete and functional test using the AXI3 Synopsys VIP models. Looking at the entire example we see that with the combination of Synopsys VIP and UVM, constrained-random verification can be performed in a well-architected and efficient manner. The protocol functionality is abstracted by the Synopsys VIP model, allowing you to work at a higher abstraction level. You can use just a small amount of code (~200 lines) to generate a very large set of conditions by controlling constraints and using test execution time instead of code development time. There is a high degree of reuse inherent in the methodology. Test-specific code is minimized with the majority of the code in the environment. Including constraints and configurations, there are only 32 lines of test-specific code!

The concepts described here will hopefully get you started with solid fundamentals and inspire you to learn more. A great next step is to visit the Verification IP web page, where among other things you can view multiple whitepapers, blogs, and webinars to help you in your journey toward the coverage closure of your SoC designs. Happy verifying! ■

About the Author

Amit Sharma, is a director in the Verification Group in Synopsys India. He is an Electronics and Communications graduate and holds an MBA from the Indian Institute of Management, Bangalore. Amit has 15 years of experience in various facets of functional verification and currently leads the AsiaPAC Corporate Applications Engineering team in deploying emulation technologies, verification IP, and other advanced verification technologies.

Addressing Physical Design Challenges in the Age of FinFET and Smarter Design

Synopsys Insight Staff

With the mainstream in IC design moving to FinFET processes and with chip architects adding evermore complex functionality to create smarter ICs, engineering teams tasked with performing physical design today have a daunting set of challenges they must address — and they need a trusted, comprehensive physical implementation tool suite like the Galaxy™ Design platform to address them.

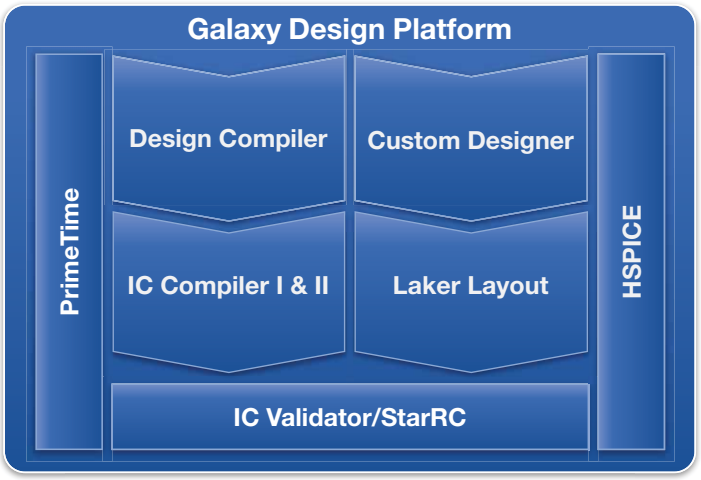


Figure 1. The Galaxy Design Platform offers a highly integrated flow of best-in-class physical design tools for the toughest designs on the most advanced silicon processes

"The most fundamental challenge for physical design teams in the age of FinFET is simply dealing with the leap in design complexity," said Mark Richards, senior technical marketing manager at Synopsys. "Design complexity is growing massively as the industry transitions from planar to what is now quickly moving to the third generation of FinFET technologies. Designs are of course bigger, have more on-chip memory, but also have more processors, more types of processors, and more complex clocking schemes to handle low power and of course higher frequencies."

Complexity is very dependent on the application. Whereas a smartphone processor might be considered on the small size these days, a GPU or server processor consists of hundreds of thousands 'step and repeat' processing units. "We've seen as many as half a billion placeable instances on an entire design," stated Richards. "We're seeing thousands of generated clocks running at very high frequencies that we need to balance by ensuring we have the lowest insertion delays," he said. "Managing all of this clock complexity is very challenging, and it's a key trend, which is why we've invested so much in developing the clock handling capabilities within IC Compiler™ II."

Increasingly, design teams have to break down big macros into small macros, which leads to fragmentation in the floorplans. Richards believes that all these challenges impact the entire design flow. “While you can shrink standard cells at each new generation, realistically when you put all these cells together, you can’t route them at the new density,” he said. “There are so many other things in flight that we need to address through place and route, modeling, improvements to routability, placement technology — there are implications for the entire flow.”

“One of the biggest challenges back-end teams face when moving from planar to FinFET is multi-patterning,” added Mary Ann White, product marketing director for the Galaxy Design Platform. “It’s an essential part of the design flow that enables foundries to print chip feature sizes that are a fraction of the wavelength of light. There are several forms. At 16- and 14-nm, double patterning was sufficient, but as we move down to 10-nm and below, self-aligned double patterning (SADP), triple patterning, and quad or more multiple patterning are essential.”

Another significant challenge is dealing with the huge increase in capacitance with 3D FinFET structures, said White. “A lot of our tools have to be updated to effectively extract these capacitances,” she said. “The challenge for the design team is how to manage the 2-3x capacitance that FinFET introduces, which can result in more dynamic power consumed. Some of the dynamic power is taken care of by moving to the lower voltage operation — from 0.9V or 0.8V to 0.5V, and in some cases lower, down to 0.35V, our tools — anchored by Design Compiler® and IC Compiler II — have several dynamic power optimization capabilities to help mitigate the effects of the extra capacitance.”

“As well as capacitive effects, resistance is going up massively node on node,” stated Richards. “The effect of grain/surface scattering at 7-nm is dominating compared to bulk resistivity, which means very high resistance, particularly in the lower metal layers. That in turn means using the multi-pattern layers is very challenging because they’re so resistive — you want to get off them as soon as possible. This one issue has implications for design speed and power, timing and accuracy. Having very resistive wires means that the waveforms going into the cell are very different from those coming out of the cell — they’re no longer linear, so the characterization becomes much more difficult.” Design Compiler Graphical and IC Compiler II have layer awareness built-in such that critical nets are promoted to upper metal layers that are much less resistive.

Transparency and Entitlement

“Synopsys aims to make the physical design issues for FinFET as transparent to users as possible,” said White. “When you move to a FinFET flow, it’s very similar to the planar-based flow that experienced users know well.”

Richards adds that FinFET processes promise a lot in terms of power, area, and performance benefits. However, whether design teams get those benefits or not when it comes to the real design really depends on the tool flow. “Our aim is to provide the entitlement part of the equation to design teams. By providing advanced flows and methodologies, we aim to get as close to fulfilling the promise as possible,” said Richards.

Two of the more important benefits of moving to FinFET include the ability to use higher transistor drive currents and to operate at a lower voltage. Higher drive current and ever narrowing wires means higher current density, which can lead to electromigration (EM) issues. Synopsys can verify and fix EM issues at the chip, cell, signal, and transistor levels. The move to a lower voltage itself brings other challenges such as process variation and waveform distortion that have to be accounted for. Synopsys’ Galaxy Design platform has several techniques — such as parametric on-chip variation (POCV) and advanced waveform propagation optimization and analysis that take care of these low voltage effects.

"Design complexity is growing massively.… Designs are of course bigger, have more on-chip memory, but also have more processors, more types of processors, and more complex clocking schemes to handle low power and of course higher frequencies."

Balancing Accuracy and Runtime

"We have to increase tool accuracy for every new node if we're going to deal with all of these engineering tradeoffs — that's a constant challenge we work to address," stated Richards.

While the Synopsys PrimeTime® tool is the gold standard for timing, Synopsys aims to align its implementation platform IC Compiler II, as closely as possible to that standard.

"Christmas always falls on December 25th, so unfortunately chip implementation schedules aren't moving to accommodate the rise in complexity," said Richards. "In practical terms, what that means is that as well as increasing accuracy, we need to maintain or even improve runtimes as instance counts grow between nodes. Developing tool flows with increased automation, accuracy, and runtime enables design teams to do more 'what-if' exploration, which leads to better quality of results."

Richards notes that another layer of complexity comes as a result of customers looking to differentiate their designs by implementing more advanced power management strategies, like selectively powering down different blocks and running in different modes of operation. Supporting the concept of "dark silicon," where design teams gate modules to slow them down or turn them off, requires the use of multiple design techniques, including clock gating and multi-voltage design. Automating such low-power design techniques is key to meeting project schedules.

Richards notes that from conception, one of the key motivations for designing and building IC Compiler II from the ground up was to dramatically improve turnaround time and capacity.

"The writing was on the wall many years ago about the key challenge for the industry: managing ever-growing design size and complexity. The vision for IC Compiler II before we started any coding was to manage high-capacity designs, minimize turnaround times, and reduce the number of design iterations required while maintaining the best possible quality of results," commented Richards. "It's all about enabling design teams to hit their market window even as designs dramatically scale in size and complexity."

Risk-Reward Tradeoffs

A key decision for chip architects nowadays is whether to make an IP block that is completely optimized for power, performance, and cost for a specific application, or to trade off that approach against implementing a more general-purpose block.

"In some ways this is a 'risk vs. reward' decision," said Richards. "If you create the best design but you miss the market for a specific application, it's likely there's nothing you can do with the chip — it's useless. What we see in some applications is design teams balancing what they commit to an IP block (so it has the lowest power possible) and what they leave programmable (so that they can adapt the design to meet evolving standards)."

"5G, or LTE, is a perfect example of this design dilemma," said Richards. "LTE is a moving target; there is no current standard as it's being developed year on year," he noted. "How you commit to silicon will impact your business. If you go for a fixed function, it isn't going to last very long so you need a lot of programmability. You could go all the way to an FPGA implementation — it's the traditional make vs. buy tradeoff — but it's becoming a more finely balanced decision and at the same time an opportunity to differentiate."

Masking Complexity

In moving to a FinFET-enabled flow, another of Synopsys' aims is to help customers become more productive by managing the physical design complexity on their behalf.

"Our job is to hide the challenges as much as possible," stated White. "Our long-established relationships with leading foundries like TSMC, Samsung Foundry, GLOBALFOUNDRIES, Intel Custom Foundry, and others enables us to work very closely with them well in advance of the processes becoming available. We identify the challenges and then figure out the best way to solve them."

A good example is routing, Richards added. "The layout engineer shouldn't have to worry about whether a route is on mask A or mask B when we're using multi-patterning, for example. We also measure our success in how evolutionary it is to move from say a 28-nm design flow to a 16-nm design flow. We don't expect engineers to have to spend six weeks re-learning how to do place and route. They can focus on the tasks that they already know how to do well — or example, importing the RTL netlist efficiently. The challenges are getting more complex, so we fully expect to continue to develop our tools to do more and more of the heavy lifting as we move to 10-nm and 7-nm process nodes."

How Synopsys manages each specific issue depends on what approach will get the best from the technology while keeping the flow easy to use.

"We maintained a colorless approach to routing at 16-nm, because it simplified the process for the end user at the routing stage," stated Richards. "We get the IC Validator tool to separate the colors later on. To the user, it looks like regular routing — they don't have to worry about the two masks. Another benefit of this approach is that there is a lot more flexibility from a routability perspective. For example, a colorless macro cell has a lot more scope to move freely, because you don't have to maintain the alignment between pins and tracks of the same color. Your design flow looks just like a 28-nm flow, and the tools make sure the design is decomposable later on in the flow — you don't have to worry about it."

However, at 10-nm, the process demands shift again and for these technologies the Synopsys flow supports colored routing in order to make life simple for the engineer while optimizing for the constraints of each process.

Accelerating Turnaround Time

The subdivided resistance and capacitance inherent to FinFET 3D transistors demands more sophisticated modeling and analysis than planar structures.

"We ensure ultimate model accuracy within the signoff tools, but for physical implementation we maintain a higher level of abstraction that enables the fastest turnaround times," commented Richards. "Our team developing the physical implementation tools works very closely with our colleagues working on the PrimeTime static analysis and StarRC™ parasitic extraction tools to ensure we take what they have and model what we need to provide the appropriate levels of accuracy with the best possible runtimes." See Figure 2.

"Synopsys aims to make the physical design issues for FinFET as transparent to users as possible…. When you move to a FinFET flow, it's very similar to the planar-based flow that experienced users know well."

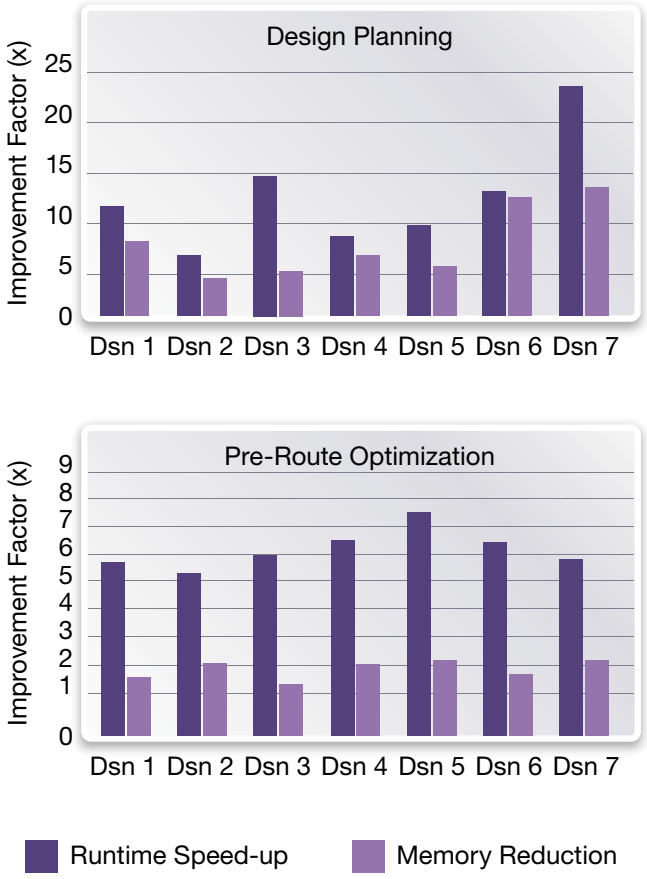


Figure 2. IC Compiler II features 10x faster throughput to accelerate product schedules on the most complex designs.

The latest modeling approach for advanced process nodes, the Liberty™ Variation Format (LVF), targets parametric on-chip variation. Design Compiler, IC Compiler II, PrimeTime®, StarRC™, and other tools within the Synopsys ecosystem look at this variation in terms of analysis and in certain cases can optimize using these kind of models. LVF uses statistical analysis to offer a less pessimistic modeling approach for advanced technologies using ultra-low voltages — particularly FinFET nodes at 10- and 7-nm and below.

Convergence to a Realizable Design

Synopsys customers are reaping the benefits of using an integrated flow, especially for more advanced processes. For example, modifying the layout after implementation affects all design targets, including timing, power, and signal integrity. In-Design physical verification provides a push-button flow for signoff-quality metal fill and design rule checking (DRC) inside IC Compiler where timing can still be considered. IC Validator In-Design physical verification combines timing awareness and signoff accuracy to speed up tapeout schedules.

"High-density current can cause electromigration problems, which is why we've designed PrimeRail to work within IC Compiler II as well," commented White. "You can do what-if analysis on IR drops and electromigration to see any potential hotspots as you manipulate the layout rather than doing the analysis as a batch process. Co-design with Custom Compiler™ allows you to do custom design on signal routes, for example, all within the IC Compiler II environment."

"We see significant benefits from having common engines across our tools for dealing with advanced processes," added Richards. "Sharing a common view of the flow across tools means that the design convergence is much better. If my synthesis tool tells me the design is fine and then I go to place and route and find it's not even routable, then I've got to go back to re-do all my synthesis. By using a shared body of knowledge to create common timing engines, common router technology, consistent models, and so on, we ensure the results out of each stage of the design flow are consistent and there are no surprises. And the bottom line is that convergence correlates directly with project timescales."

An added advantage of the Synopsys Galaxy flow is that leading foundries use Synopsys' Technology Computer-Aided Design (TCAD) tools to develop and optimize their semiconductor processing technologies and devices.

"Having TCAD within the Synopsys portfolio is a bit like having a cousin who knows someone in the band," stated Richards. "You get to hear the latest music before anyone else does. Through TCAD we had awareness of FinFET technology probably 10 years before customers were designing with it and are able to align our tools with the processes. Having the inside track through close ties with foundries — and academia — is vital if we're to keep our customers hitting the ground running with the latest technology." ■

To learn more about the Galaxy Design Platform for implementation, click here!

Multi-Level Physical Hierarchy Floorplanning vs. Two-Level Floorplanning

Steve Kister, Technical Marketing Manager, Synopsys

Today's SoC designs require hierarchical layout methodologies that span multiple levels of physical hierarchy. Many of today's physical design tools only handle two levels of physical hierarchy at a given time, limiting potential quality of results (QoR) that design teams can achieve in the time they have scheduled for layout. Synopsys' IC Compiler™ II provides automation for designs with multiple levels of hierarchy. This minimizes time to results, provides best QoR, and maximizes productivity of physical design teams.

Let's look at the challenges inherent with tools limited to two levels of hierarchy, discuss why design teams need multi-level physical hierarchy floorplanning, and explore how IC Compiler II addresses the challenges of floorplanning modern SoCs.

The Challenges of Floorplanning Large, Complex SoCs

The operating speeds and capabilities of large, complex SoC designs are striking, and the task of physically implementing these designs is daunting. Today, design teams are dealing with 100s of millions of gates — tens of millions of placeable instances. To manage this complexity, physical design teams use hierarchical implementation methodologies to divide a single chip into sub-chips. Often the sub-chips are so big and complex that they too must be divided into even smaller sub-chips (Figure 1). Using three, four, or even five levels of physical hierarchy to implement a design is becoming the norm.

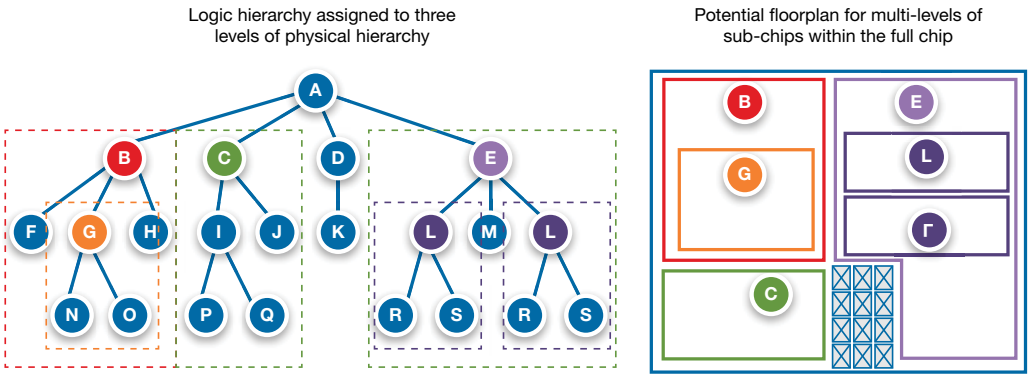


Figure 1. Physical floorplan divided into three levels of sub-chips

Design teams floorplanning today's largest and most complex SoCs are finding it increasingly difficult to perform and manage physical planning with tools limited to two levels (top and block) of physical hierarchy. A two-level-at-a-time physical implementation flow requires designers to employ a recursive flow. The added flow complexity and data management impacts the productivity of design teams and delays design schedules. Truly supporting multi-level physical hierarchal planning and layout is an opportunity for planning, place, and route tools to improve designer productivity and efficiency, and to minimize design schedules.

Pitfalls of Recursive, Two-Level-At-a-Time Hierarchical Methodologies

With a two-level-at-a-time physical implementation flow, design teams are forced to set block shapes, set pin placement for each block, and budget block timing constraints one level at a time. This makes it very hard for design teams to plan all of the lower levels within the context of the full chip design. Instead, they are forced to plan lower levels in the context of constraints defined on each level's parent level parent (Figure 2).

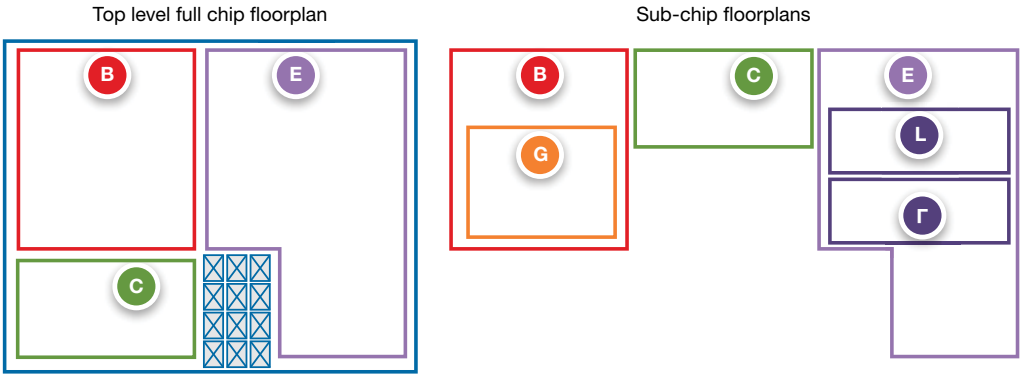


Figure 2. Multi-level physical floorplan divided into several two-level (top and block) floorplans

The shapes, pin placements, and budgets set at a given parent level become hard constraints for the child level of the design. If the design team makes a change to one of the blocks at a lower level, that change can impact the top level of the design. To address such a change, teams have to consider how a given change will impact other levels and what extra work will be required to make those changes. Simply deriving a plan to implement one of these changes takes a lot of negotiation between design team members and can be quite complex. The exercise can often lead to schedule slippage.

Indeed, design teams employing a two-level-at-a-time physical implementation flow spend a significant amount of their overall design time managing data and tracking tasks. For example, to manage design complexity and minimize memory use, teams will convert the children of a given level into black boxes—even if they have access to the netlist data. They often boil down area requirements to an educated engineering guess and must manually account for any constraints on shapes or sizes to accommodate hard macros within the child block. Teams take ownership of creating black-box models and the subsequent tasks of replacing black-box models with real netlist data at later points in the flow.

There is the potential for designers working at the child level to create and compound problems unknowingly. Consider two designers each working on a sub-chip that is connected through logic at a parent level. Each designer may choose to move content within their sub-chip to improve routability and timing. However, when they put these into the context of the full chip, the designers may have created timing paths that cannot be closed easily (Figure 3).

"Using three, four, or even five levels of physical hierarchy to implement a design is becoming the norm."

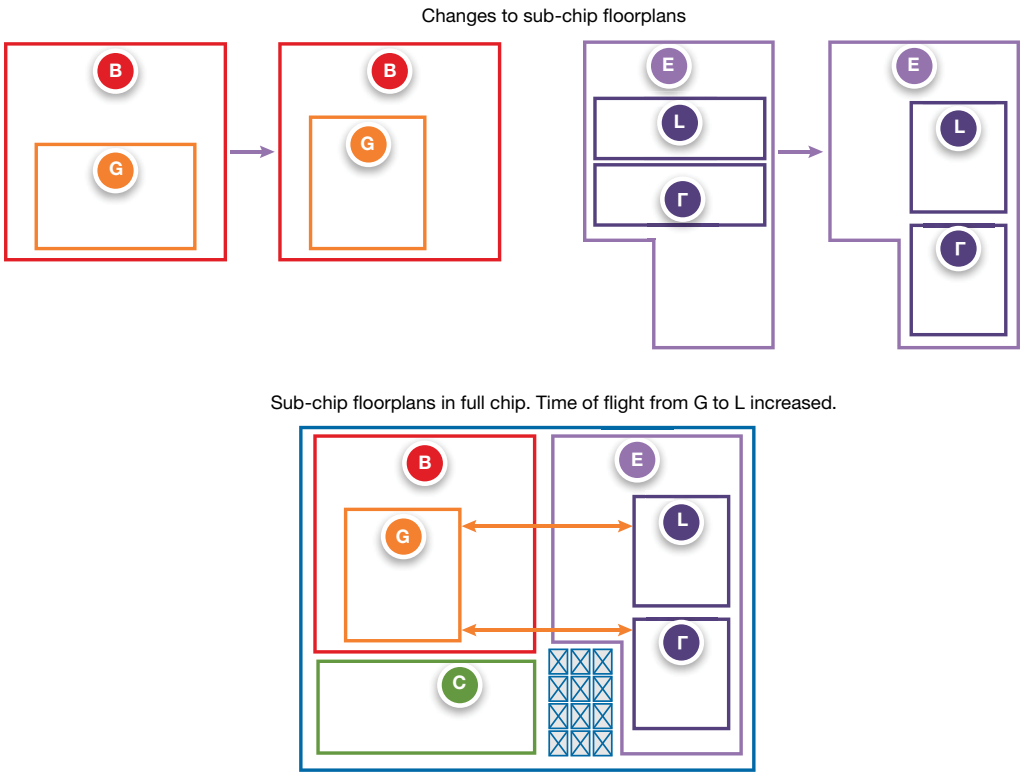


Figure 3. Example of sub-chip changes that can make top-level closure more difficult

Another level of complexity is introduced when teams use a multiply instantiated block (MIB). An MIB is a physical block that design teams intend to use multiple times in a given design. Design teams creating a given MIB must consider where and how they will use the MIB throughout their layout. For example, when design teams intend to use MIBs within lower level sub-chips, they must consider the requirements for each unique environment when defining the shape, size, pin placements, and timing constraints needed to implement the MIB. Adding to the complexity of multi-level physical hierarchy planning, MIBs may be instantiated at any hierarchical level. So, once again, design team members need to spend cycles planning where to use MIBs within multiple sub-chips.

These pitfalls and others are the source of wasted time and lost QoR in designs. Now let's explore a better way of dealing with the hierarchical complexities of today's designs.

IC Compiler II Provides Multi-Level Physical Hierarchy Planning

Synopsys IC Compiler II features a new data model with native physical hierarchy. Most physical design tools were created with a flat database and physical hierarchy was added later as a quick competitive response. However, native physical hierarchy provides significant advantages for multi-level physical hierarchy planning and implementation. All engines—shaping, placement, routing, and timing to name a few—quickly access the specific data relative to physical hierarchy that they need to perform their function.

For example, consider shaping. In Figure 4, the shaper needs to know the target area for each sub-chip, aspect ratio constraints dictated by hard macro children, and interconnect that exists at sibling-to-sibling, parent-to-child, and child-to parent interfaces. If the design is a multi-voltage design, the shaper needs target areas for voltage areas. These needs add more constraints for the shaper to deal with in the design. For multi-level physical hierarchy planning, shaping constraints that exist for lower level sub-chips translate up the hierarchy in the form of shaping constraints on parent sub-chips. The shaper does not need

to know about the full netlist content that exists within each sub-chip or block. Synopsys constructed IC Compiler II's database with this idea in mind, providing engines with the specific data required for their particular function. For multi-voltage designs, IC Compiler II reads UPF and stores appropriate data within sub-chip levels. The engines pull data from the database to calculate targets based on natural design use or pull user-defined attributes specifying targets.

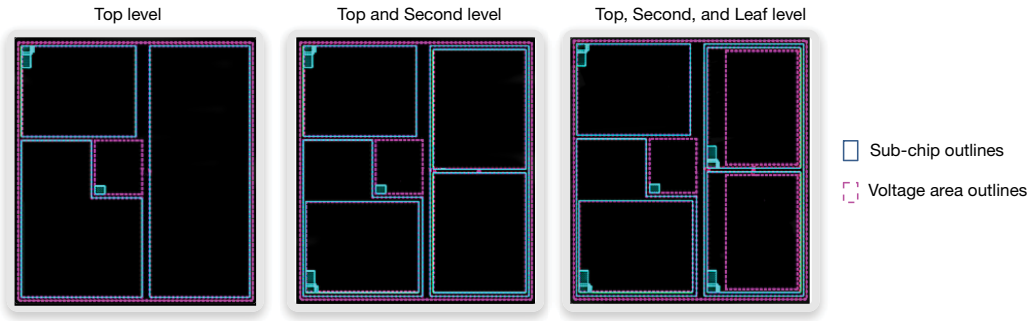


Figure 4. IC Compiler II multi-level shaping results

Synopsys built the new data model to support distributed processing architectures. IC Compiler II engines split their jobs, or parts of their jobs, into multiple processes. As shown in Figure 5, cell placement is a good example of this synergy.

After shaping, the placement engine sees a global view of data flow interconnect paths at physical hierarchy boundaries and connectivity to macro cells. With this information, macros are placed for each sub-chip at each level. Understanding relative location requirements of interconnect paths at boundaries ensures there are resources available at the adjacent sub-chip edges to accommodate interconnect paths. The placer anticipates the needs of hierarchical pin placement and places macros where interconnect paths do not require significant buffering to drive signals across macros.

Using sub-chip shapes, locations, and the global macro placements, the placer engine models the external environment seen at the boundaries of child and parent sub-chips. Using the model, the placer creates cell placement jobs for each sub-chip at each level of hierarchy. Each job creates placement for standard cells of each sub-chip. By splitting this into multiple processes for sub-chips, IC Compiler II minimizes turnaround time while maximizing the use of compute resources.

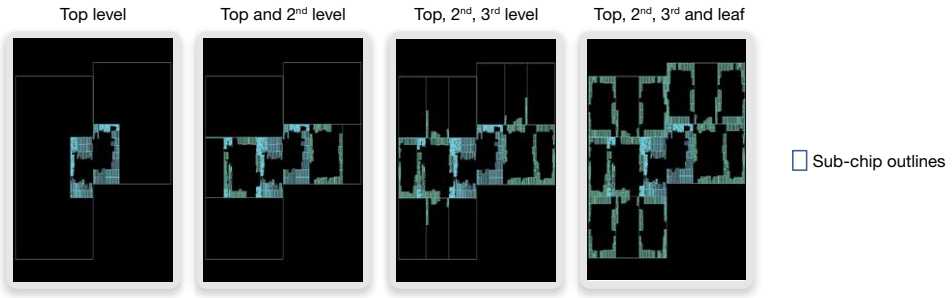


Figure 5. IC Compiler II multi-level global macro placement results

For power routing, IC Compiler II provides an innovative object-based methodology. The tool takes patterns describing construction rules such as widths, layers, and pitches required to form rings and applies meshes to areas based on floorplan objects, such as voltage areas and groups of macros. Strategies associate patterns or multiple patterns with areas.

"Design teams creating a given MIB must consider where and how they will use the MIB throughout their layout."

A design team can set up a complete set of strategies for the full chip. Given these strategy definitions, IC Compiler II characterizes the power plan and automatically generates definitions of strategies for sub-chips at all levels and generates a complete power plan in a distributed manner. It is worthwhile to note that sub-chip design teams can use these strategies as needed. Because the characterized strategies are written in terms of objects at each sub-chip level, design teams can easily use the tool to recreate power plans to accommodate floorplan changes at any level.

With shapes formed, macros placed, and power routed, IC Compiler II's pin placement engine pulls interface data at all levels and invokes a specialized global router to determine where to place hierarchical pins. Synopsys gave the specialized global router the ability to recognize physical boundaries at all levels to ensure efficient use of resources at hierarchical pin interfaces. The tool will align pins across multiple levels when possible. The specialized global router is MIB-aware — like all IC Compiler II engines — to guarantee it intersects with the edges of MIBs identically.

To place pins for MIBs, the tool's pin placement algorithm determines the best pin placement that works for all instances, ensuring that the pin placement on each instance is identical. Additionally, pin placement creates feedthroughs for all sub-chips, including MIBs, throughout the hierarchy (Figure 6). The global router engine plans feedthroughs across MIBs and determines feedthrough reuse and tie-offs of unused feedthroughs.

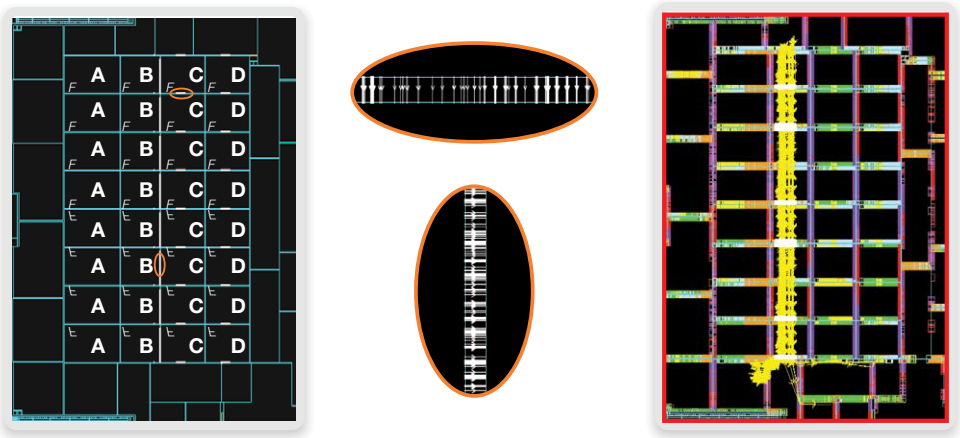


Figure 6. IC Compiler II pin assignment and feedthrough creation results

Once IC Compiler II places the pins, the tool estimates the optimized timing at hierarchical interfaces and creates timing budgets for sub-chips. Most EDA tools' budgeters create timing constraints that are applied to primary hierarchical input and output (I/O) pins of sub-chips. The (I/O) delays of the budgeted constraints represent timing path segments that exist in the parent of sub-chips. This enables sub-chip designers to perform placement and optimization of "flat" sub-chips by modeling the external timing environment seen at their primary I/O pins. However, this does nothing to model the internal timing environments of sub-chip pins as seen at their parent levels. There are no constraints to represent timing path segments that exist in the child sub-chips. IC Compiler II's advanced budgeter creates timing constraints for all child interface pins within the full chip, the parent, and child interfaces for mid-level sub-chips and the primary pins at the lowest level sub-chips. The entire design can proceed with placement and optimization concurrently and in a distributed manner.

As the design matures and various sub-chips are closed, teams are able to specify sets of sub-chips that can and cannot be physically modified. This helps when design teams are required to update current netlist data with new netlist drops. Consider a case where the new netlist data requires an increase in a sub-chip size to accommodate growth in logic produced by synthesis. IC Compiler II incrementally updates the floorplan, considering the full-chip context and the sets of sub-chips that can and cannot be modified.

This minimizes the amount of wasted design time required to negotiate changes that ripple through to different levels of the design hierarchy. At any point in the planning or implementation flow, design teams can interactively view, analyze, and manually edit any level of the design in a full-chip context. For example, a team may choose to hand route a timing-critical signal that passes through multiple levels of the hierarchy. In IC Compiler II, they can open the full design in a layout editing window. Users can choose to view the top level only, or multiple levels of hierarchy.

When viewing multiple levels, design teams can perform interactive routing as if the design is flat. When completed, IC Compiler II automatically pushes routes into children and automatically adds hierarchical pins (Figure 7).

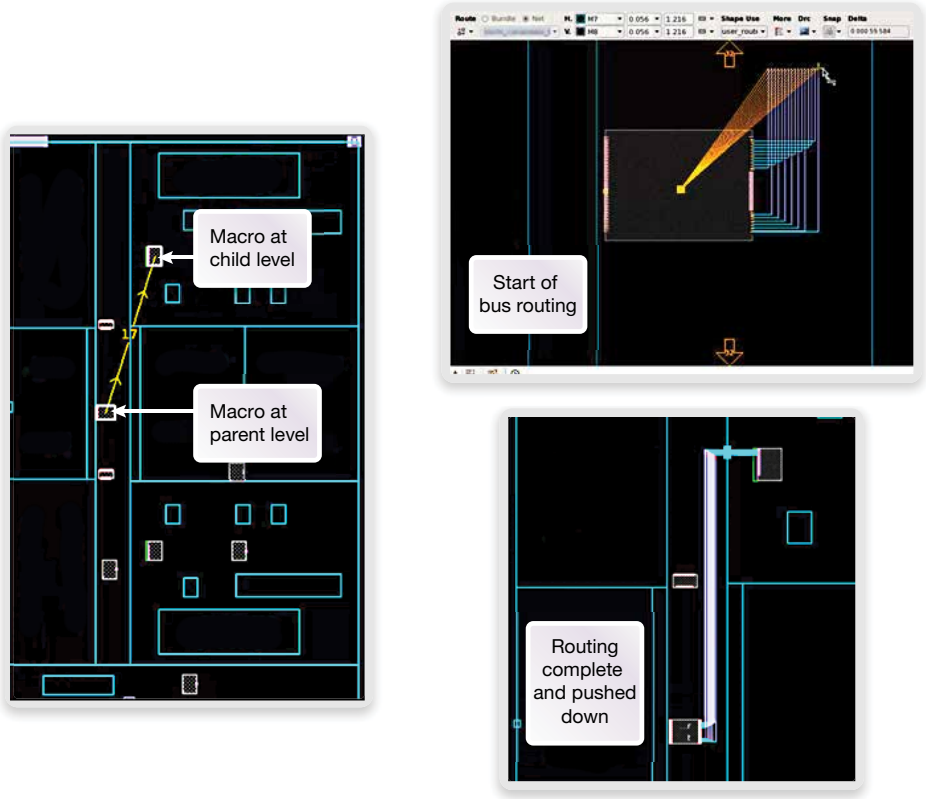


Figure 7. Interactive multi-level route editing

Today's designs require multi-level physical hierarchy planning and implementation methodologies. However, EDA tools that are limited to two levels of physical hierarchy make the multi-level hierarchical planning task difficult to implement, often causing teams to compromise design area, functionality, and QoR in order to adhere to tight tapeout schedules. Synopsys built IC Compiler II with a new data model that provides the infrastructure required to automate multi-level physical hierarchy floorplanning. With this automation, IC Compiler II provides design teams with the best QoR and time to results. ■

For more information, click here!

About the Author

Steve Kister, is a technical marketing manager supporting place-and-route design planning tools in Synopsys' Design Group. Steve has been in the ASIC design and EDA industry for 35 years, having spent the past 20 years at Synopsys. Steve received a BSEET from the DeVry Institute of Technology, Phoenix, AZ.

IP for the Era of FinFET and Smart Designs

Synopsys Insight Staff

With IC foundries actively developing the third generation of FinFET processes, it's clear that FinFET presents some very respectable technological advantages over planar processes. But developing IP for the FinFET node and integrating IP in designs targeted for FinFET processes also presents new challenges that designers must consider. Working closely with foundries helps Synopsys develop the IP that SoC designers need to get the most out of the latest FinFET processes and bring to market smarter SoCs that increasingly incorporate multi-processing.

Navraj Nandra, senior director of marketing for DesignWare® IP products at Synopsys, said that FinFET offers many of the expected new process technology advantages over planar technologies, fulfilling the top three design benefits — better power, performance, and area — of new process technologies.

"The FinFET concept uses a very thin channel that can be fully depleted of carriers, which gives much better electrical characteristics," said Nandra. "You can better control the short channel effects and reduce some of the horrible things that are happening at 20-nm planar. FinFET offers lower leakage and lower voltage because of the reduced drain-induced barrier lowering and better sub-threshold slope."

Nandra said that by working closely with IC manufacturers, Synopsys has now completed more than 60 test chip tapeouts across all the leading companies, providing "proof points" for FinFET IP on three generations of technology: 22-nm, 16/14-nm, and 10/7-nm processes. The FinFET designs do indeed achieve consistently better power, performance, and area than 28-nm and 20-nm planar processes (Figure 1).

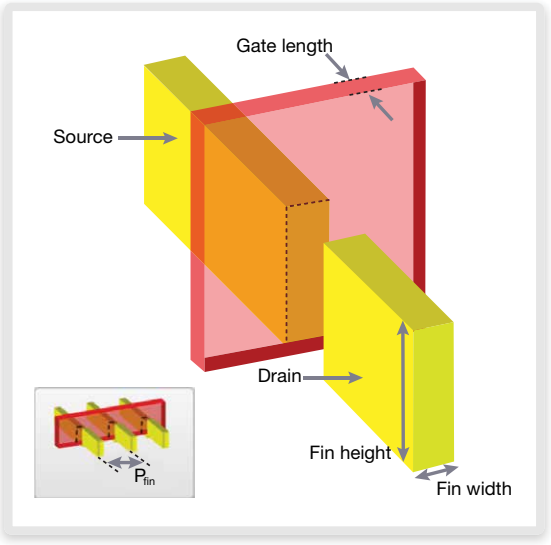


Figure 1. FinFET technology delivers lower leakage, less variability, and lower voltage.

Shaping Performance and Area Efficiency

By working with foundries for three generations of FinFET, Synopsys has learned how to get optimal performance and area from the new processes. "Maintaining the ratio of FinFET channel length to fin width at 2.5 prevents short-channel effects," said Nandra. "The fabs have increased the current per layout area by making the fins taller, which boosts performance and provides area benefits. Another technique called fin depopulation — removing the fin after you make it taller — also helps with performance through smaller area and shorter interconnect."

Fin depopulation reduces the dynamic power significantly, as well as increases speed and improves the overall speed-power metrics power-delay product (PDP). "There are other interesting things happening," Nandra said. "Layout engineers are starting to use routing optimization with air spaces to improve PDP. Foundries can also trade off the electrostatics to increase speed or reduce power, so now you can have a low-power version and a high-performance version of that latest generation of FinFET technology."

Foundation IP

Synopsys is using its experience in designing with FinFET to create high-speed and ultra-high-density foundation IP that takes advantage of the unique properties that the technology offers. "By designing special memory compilers and logic libraries — we call it HPC or high-performance core design — we can get very good power/performance/area on 7-nm devices," said Nandra. "The specialized cells are optimized for CPUs, GPUs, and DSPs, and include high-speed flops, special clock drivers, metastable optimized flops, and many other cells optimized for 7nm (Figure 2). For example, with FinFET, design teams are pushing down core voltages to a point where devices start to not work. To support this approach, we have built read- and write-assist technology that can support a minimum voltage lower than the foundry's rated voltage. Without write assist, we see about 230 fails in 10,000 Monte Carlo runs with a 0.63-volt supply. With the write-assist technology, we see no fails at 630 mV using the worst-case PVT. This is a good example of how we're taking 7-nm technology and doing more to it to enhance the value of the foundation IP blocks."

Component	CPU	GPU	DSP	Primary Benefit
High-speed flops	●		●	Critical path performance
Area-optimized flops		●	●	Register area
Multi-bit flops		● ●	● ●	Dynamic power/area savings
Multi-bit pulsed latches		● ●	● ●	Dynamic power/area savings
Duel edge triggered flops		● ● ●	● ● ●	Speed, dynamic power/area savings
Secial clock drivers and ICGs	● ● ●	● ● ●	● ● ●	Speed, dynamic power/area savings
Delay cells	● ● ●	● ● ●	● ● ●	Accurate hold fixing/area
Metastable-optimized flops	●		●	Synchronizers
High speed memory instances	●		●	CPU caches
High density memory instances		● ●	● ●	Area and power optimized instances

Figure 2: 7-nm specialized cells for CPU, GPU, and DSP cores

The latest processes introduce new defect elements that are specific to 7nm. Alongside the generic process variation systemic random faults, the fin height creates width variability that has to be modeled for embedded memories. Furthermore, the resistance and parasitic line capacitances become critical at 7nm and have to be modeled in the fault repair technology.

"We have implemented new memory test algorithms that enable us to ramp 7-nm technology with performance and yield," said Nandra. "That's only possible because we have access to the FinFET devices."

"By designing special memory compilers and logic libraries…we can get very good power/performance/area on 7-nm devices."

"We have implemented new memory test algorithms that enable us to ramp 7-nm technology with performance and yield".

The Impact on IP At 7nm

Nandra said that at 7nm, customers are using a broad range of clock frequencies. "On the one hand we see customers pushing the envelope with frequencies exceeding 2 GHz," said Nandra. "We've also seen an uptick for designs at the lowest "beat'— around the 50-200 MHz mark, which is explained by the surge of products for Internet of Things (IoT) applications." Synopsys' latest survey of more than 500 customer designs shows that many design teams now employ more than 20 clock domains, and seven percent incorporate more than 1,000 clock domains.

"At the top end of clock speeds, we've reached some fundamental limits," said Mike Thompson, senior manager of product marketing for ARC processors at Synopsys. "Specifically, using faster memory access times to try to reach very high speeds just generates too much power and heat. We're limited to realistic frequency ranges of 2-3 GHz. Even 2 GHz can be a challenge in an embedded application because of the power consumption and having to get rid of the heat."

Nandra added that when designs incorporate several clocks and many clock domains, design teams need to balance them across their designs. "With these advanced technology nodes there are parameter variations across the chip, which make it virtually impossible to control delays — especially in the clock networks and any other global signals," said Nandra. "To manage the increase in routing delays, we're seeing design teams adopt a technique that's probably 25 years old — globally asynchronous locally synchronous (GALS) design — for complete chips and large IP blocks. The clock net power is very dominant; you've got to manage the clock skew over multiple corners and then perform asynchronous data exchange. The GALS technique uses dedicated synchronous blocks and handshaking asynchronously between them."

"Of course, there's another 'gotcha' with clock speeds," added Thompson. "While logic speed increases with each new node, the memory speed doesn't increase all that much. In 28nm, you can access a DRAM at about 1.4 GHz. That means you're limited to about a gigahertz for single-cycle access to memory to be able to access the memory, read the contents, and get the information back. As we move to smaller FinFET processes, the logic is even faster, but we're constrained by the memories. To counter this, our high-speed processor IP architecture supports two-cycle addressing, which enables memories to run at half the speed of the logic. This means you can clock the processor much faster than the memory. It has a dual effect: first of all, it allows design teams to access the memories that are available, and in many cases use higher density memories rather than the higher speed memories."

Enhancing Reliability

Clock skew is just one example of how the variability of 7-nm technology impacts digital design. New approaches to timing constraints and RAS (reliability, accessibility, serviceability) are also required to counter the process variability, lower supply voltages, and higher clock frequencies.

"Teams are designing so much redundancy and replication into their RTL for advanced nodes that some of the consumer products are beginning to look like they're intended for defense markets," said Nandra.

"Our IP team works closely with the EDA team to share the challenges in IP development. Ultimately this helps enhance what's going on in the EDA world. As a result, you can add — through the tools — features such as triple model redundancy and three-distance state machines and preserve these in the back-end flow."

Multi-Core Processors for Smarter SoCs

As the FinFET processes present new challenges for design teams, they also enable companies to place more functions on their designs. And in the era of smarter design, that often means more types of processors on one SoC. Thompson notes that the mix of processors will change dramatically depending on the targeted end market.

"We see a world of difference between the needs of processors for servers that enable the cloud and the IoT devices at the cloud edge," said Thompson. "Design teams are building huge multi-core structures to do very wide parallel processing. Integrating 24 processor cores within a network processor is not unusual these days. Multi-processing in an edge device is less common. Where there's a need, it's usually for heterogeneous architectures where the control and signal processing is implemented on separate cores."

Thompson sees multi-core as a key solution to finite clock speeds because of power and memory constraints. "Another thing we can do is become more efficient in terms of the processing capability of a processor itself," he said. "In other words, the raw instruction performance, or how much work you can do per instruction. We've worked a lot on that both at the high (ARC HS Processor — 10-stage pipeline) and low (ARC EM Processor — 3-stage pipeline) ends of our processor capabilities. We've increased efficiency so that we can do more with the processor at a given clock frequency, but we can also do more with less memory, which is very important for the more deeply embedded IoT designs."

"We're getting some interesting requests now where customers want to put the ARC EM Processor with HS Processor in a multi-core structure, to do different tasks obviously, but they want L1 coherency across that structure or even L2 cache support. We've been seeing this trend for about a year, and I believe that it's process-related. As we move from 28nm, everything gets smaller, but there's a desire to push the power down by even more than is possible by relying on the technology shrink alone. With innovative multi-core structures, you can do that."

Thompson has also noticed an uptick in the variety of designs that are power constrained. "We're even seeing design in automotive applications becoming highly power constrained," said Thompson. "You'd think there would be an abundance of power generated by the engine, but there are limits. Increasingly, the system architecture in cars is defined by wanting to locate the high-end processing close to the source of the input. For example, putting an HD vision processor with the forward-looking camera on the windshield. The HD camera generates a huge amount of data, and processing the data at the camera dramatically cuts down on the amount of data that has to be moved around the vehicle. When moving large amounts of data around the car, there are power issues…there's the potential for corruption — the car is a noisy environment. While keeping the data close to the camera is ideal, it means you've got to put a sensor and a high-performance processor on the windshield to deal with the high-resolution data. Of necessity, we have to look very closely at power consumption of these cores to make them very efficient."

Real-time, high-definition vision processing requires very high bandwidth to support 2 megapixel images with 12-16 bits per pixel at a minimum of 30 frames per second. Synopsys' processor R&D team has created the EV6x vision processor family that allows design teams to address HD vision resolutions very efficiently.

"Design teams are building huge multi-core structures to do very wide parallel processing. Integrating 24 processor cores within a network processor is not unusual these days."

"We've combined our high-performance 32-bit scalar pipeline with a 512-bit wide vector DSP to create a vision CPU specifically for HD vision applications," Thompson said. "We can configure up to four of those in a single processor and even add a dedicated convolution neural network (CNN) processing engine, which is extremely efficient. It's completely programmable, but its capability is very close to dedicated hardware in terms of power consumption. This gives us a very sophisticated structure — one of the most capable processors I've worked on in almost 35 years" (Figure 3).

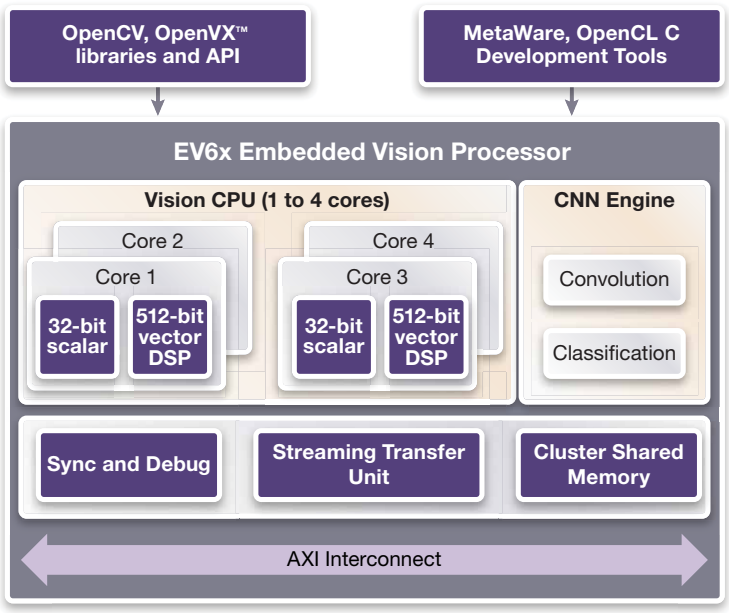


Figure 3. DesignWare EV6x processor block diagram

Transitioning high-performance vision processing to embedded applications requires the use of specialized processors that offer many of the capabilities and bandwidth of those in a laptop, but at a fraction of the power.

"Moving to multi-processor structures will help us address these physical limits, but we still have to do that cost-effectively," said Thompson. "We can't ignore memory, which can blow up very rapidly. Take Microsoft Windows, for example. I think version 3.11 took a couple of megabytes. Windows 10 is about 10 gigabytes. Of course Windows 10 offers users a lot more capability, but that's pretty typical of what happens with applications — we have to find ways to keep the memory size reasonable when we embed applications because it burns power. It's all part of the power-performance paradox."

Software for Multi-Core

Thompson believes that heterogeneous structures like the EV6x vision processor family, which consist of dedicated programmable functions seamlessly attached to other processing capabilities, offer a way forward for embedded design.

"Programming heterogeneous structures is more complex than single processors, but software support is much better than it used to be," he said. "The latest programming tools allow software engineers to create kernels that they can direct to any resource that's available in the hardware. If you write a kernel in Open CL, which is the programming language for wide vector structures, you can apply that kernel to the Vector DSP while directing kernels written in C/C++ to a scalar unit and so on. That gives the programmer a level of abstraction from the hardware that's very convenient — they don't need to be that tied in to the details of the underlying hardware; they just know it's there."

"Software development is around 60-80% of the cost in delivering the application. That's going to continue to increase," continued Thompson. Assembling the underlying hardware using processor and interface blocks can be very quick, but verifying everything and writing the software — all the stacks and drivers — that can take a great deal of time."

Thompson notes that a lot of the R&D effort at Synopsys today is focused on designing tools that enable customers to develop software well ahead of the availability of silicon. "Virtual prototyping allows software teams to write and debug code before hardware is available," he said. "Having a strong ecosystem that supports teams with solutions for developing code is essential. We have focused on the software tools and open development systems through embARC Open Software Platform. Giving developers access to proven tools and software IP helps them to manage their schedules more predictably."

To complement technology scaling and processor architectures, design teams are also innovating with packaging technologies, said Nandra. "They are taking designs done in a previous-generation process, keeping the design as is, and then using interposer technology to connect to the high-performance die that's created in the 7-nm process," said Nandra. "Using technologies like high-bandwidth memories, interposer technology, 3-D transistors, and 2.5-D packaging enables design teams to leverage a common substrate with two different die. It's this holistic, multi-variant approach that encapsulates IP design for the FinFET era." ■

"Software development is around 60-80% of the cost in delivering the application. That's going to continue to increase."

From Silicon to Software: A Quick Guide to Securing IoT Edge Devices

Ruud Derwig, Senior Staff Engineer, Synopsys

The Internet of Things (IoT) is undeniably a hot topic right now. There are many definitions of what an IoT device is exactly and debate about what they will be able to do and how they will change our lives in the future. Regardless of the type of IoT device or application, one of the requirements that seems to be universal for all IoT devices is the need for security. Although it includes functional aspects, security is an emerging system property that design teams cannot simply accomplish by integrating a single, magic security IP block into an SoC or by embedding a magic algorithm in the software stack of every IoT system. IoT architects must look at security holistically and make complex tradeoffs that are interdependent and span multiple disciplines. Concurrently, they must also balance hardware versus software, throughput versus area and energy consumption, and security level versus cost. For brevity's sake we will focus here specifically on IoT edge nodes.

System-Level Protection Mechanisms

When designing IoT edge devices, architects must make key architectural decisions at the system, processor, and software levels. For example, at the system level, the main decisions include (1) what to implement in hardware and what to implement in software; and (2) how to guarantee strict separation of secure system resources. It's also helpful for the hardware and software teams to understand the security mechanisms the others are employing or the options available to the team, as each level of security comes with some degree of overhead in terms of complexity, design time, and cost.

The first hardware-software tradeoff is the implementation of the foundation functions like cryptography algorithms. We can implement them fully in software, use hardware to accelerate software, fully implement them in hardware, or use a combination of those options. Depending on system requirements for throughput and latency on the one hand, and silicon area and power efficiency on the other hand, we can choose which approach is best for our design. Figure 1 shows the relative differences between three options for the SHA-256 secure hash function. As a baseline (shown on the left), we'll use a software implementation on a DesignWare® ARC® EM processor. In the middle is an approach that uses hardware to accelerate software known as CryptoPack, an option for the ARC EM processor that adds new instructions through ARC Processor EXtension (APEX) technology, which accelerates cryptography processing. Synopsys designed CryptoPack to achieve significant performance gains at a modest area increase.

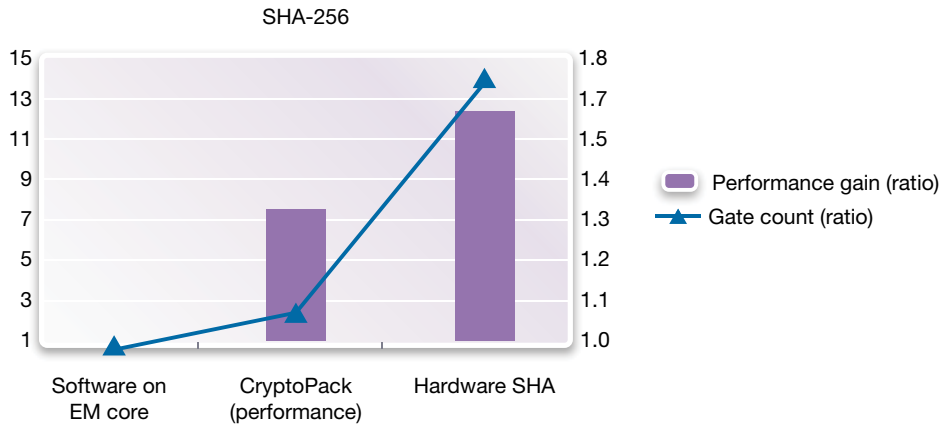


Figure 1. Tradeoffs between software and hardware crypto implementations

For the SHA-256 algorithm, the CryptoPack implementation is more than seven times faster than a plain software implementation and adds less than 10 percent to the gate count of the baseline ARC EM processor. If the system requires higher performance, we can use a full hardware crypto engine. The hardware SHA implementation on the right in Figure 1 uses a DesignWare security IP core that almost doubles performance over the CryptoPack version but at a higher area increase—almost 70 percent more gates are required. Similar to the security foundation functions, we can make a similar tradeoff for other software like higher level communication protocols. If the design requires even higher throughput, or if the system requires we free up even more processor resources by off-loading parts of the protocol processing, we can use hardware protocol acceleration from DesignWare security protocol processors.

After adding up all the required CPU cycles for both security-related software and other software that needs to be executed for a specific use-case, it becomes clear how many processing resources are required in the system. For example, for a simple IoT edge node, typically a single core is adequate. But for a system with more functionality, it could be beneficial to use multiple cores to reduce total system power consumption by running the chip at a lower frequency and voltage operating point. The other consideration for planning the number of cores in a system is to guarantee strict separation of secure system resources.

The simplest and most traditional way to separate trusted, secure software from normal, non-trusted software is to use multiple cores and allocate software to a core according to its security profile (left-hand side, Figure 2). A general-purpose CPU runs — optionally on top of a small Real-Time Operating System (RTOS) — normal application software that is not specifically trusted and that could come from any source, including end-user programming. For security-related software, we can use a second, secure CPU running its own software stack fully isolated from the other processor. Since it does not share resources with the other processor, there is no risk of leaking secrets to or having unauthorized modifications made by the general-purpose CPU. This does require that we physically separate the memory for each CPU by tightly coupling the memory to the processor using separate buses or by using other bus and address map protection mechanisms, as will be explained later. An inherent drawback of this strict hardware isolation of normal and secure worlds is that communication between both worlds can become more complex. We would either have to add a shared resource like a small shared memory or would need to add a dedicated communication channel that connects both CPUs. If special hardware is available for inter-processor communication, like ARConnect for ARC processors, we can use this for secure communication between both cores in our designs.

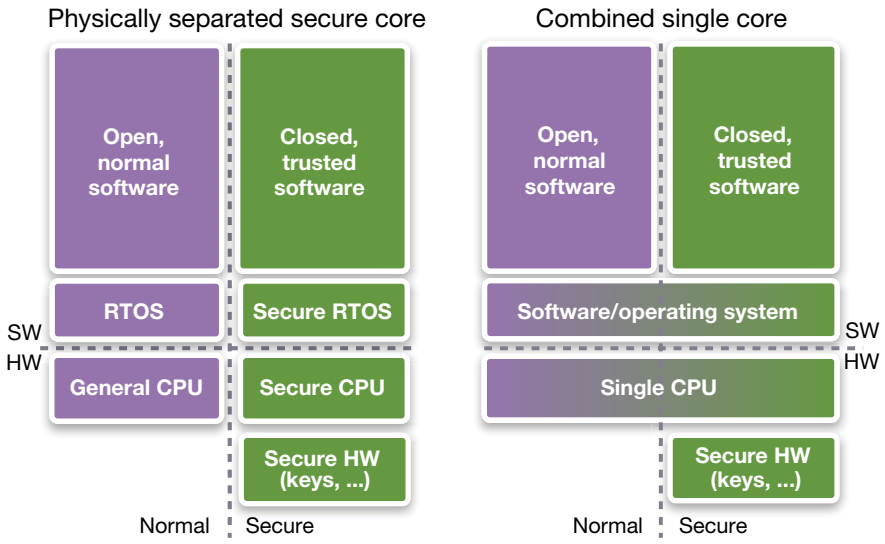


Figure 2. System processor partitioning options

Alternatively, as depicted in the right of Figure 2, we can combine both trusted and normal software on a single core. In this case, however, we must implement mechanisms to isolate both worlds. There are several well-known technologies for such separation. A software-only solution, for example, is to run the trusted software inside a virtual machine. For better performance, instead of using a software-implemented virtual machine, we can use processor hardware to switch between multiple execution contexts and control access to system resources for each of these contexts. Such protection can be provided by CPUs in the form of multiple privilege levels and a memory protection unit (MPU), as in the case of ARC SecureShield™ technology.

The bottom right of both sides in Figure 2 includes a box representing secure, non-CPU system resources that require protection. These resources could include secret keys but also specific peripherals like a fingerprint reader. There are several ways of protecting access to hardware resources. For traditional designs using a bus-based infrastructure, we could use either multiple isolated buses (secure and non-secure), or we can add a sideband signal to the bus that indicates whether the bus transaction is considered secure or not. Figure 3 schematically depicts such a scenario. Only when the “secure” sideband signal is active, access to slaves like memory and peripherals is granted. The sideband signal is driven by the master initiating the request and should be trusted. This means not only that a secure CPU processor must truthfully generate this signal, but also that all other masters on the bus (like a normal, open CPU or a DMA-capable peripheral controller that is programmable by non-trusted software) should implement these signals.

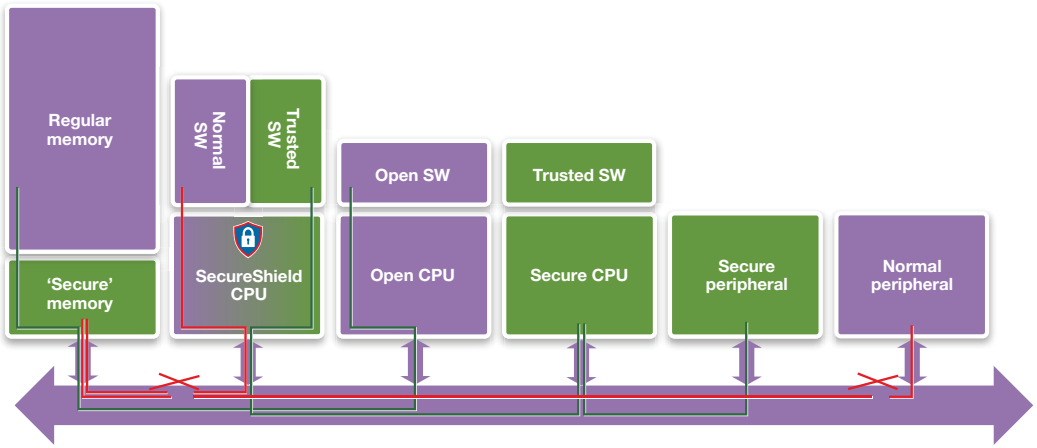


Figure 3. Access control with secure bus sideband signal

An alternative way of protecting secure hardware resources is to not access them through a public, shared bus, but rather to access them directly from the CPU. This "close coupling" is a feature that was initially developed for saving area and power. By leveraging the extensibility of a CPU like ARC with APEX, instead of accessing the control registers of a peripheral controller via a regular bus, we can map the control registers directly to the register space of the processor. The already existing mechanism of CPU privilege levels controls access to the peripherals in this case. Figure 4 below depicts such a system.

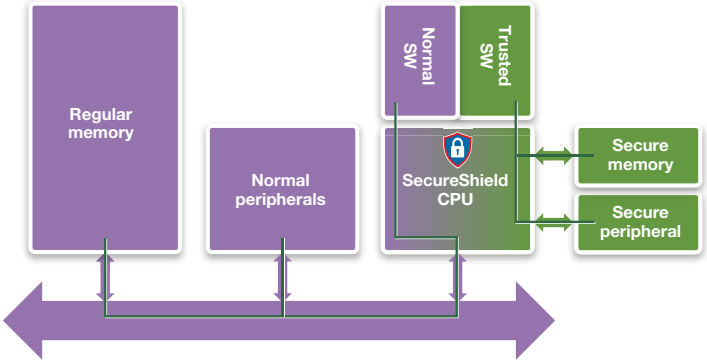


Figure 4. Access control with tightly coupled memory and peripherals

Besides protecting non-trusted access to peripherals, we can also use this technology for secure memory accesses as well when the CPU supports such close coupling for instruction and data memory.

"The simplest and most traditional way to separate trusted, secure software from normal, non-trusted software is to use multiple cores and allocate software to a core according to its security profile."

Processor-Level Protection Mechanisms

Beyond supplying mechanisms for separating secure and non-secure functions as described above, we can enhance a processor with several security features to protect against tampering, fault detection and side-channel attacks, as well as prevent reverse engineering and IP theft.

To support tamper and fault detection, we can activate features in the processor for checking the integrity of memory and processor registers. Using parity bits and Hamming codes, we can detect bits that flipped due to tampering using, for example, LED or laser flashing. We can raise the processor's secure execution privilege level to its highest level, so that if someone tampers with the memory or registers, a trusted exception handler can take adequate countermeasures. Depending on the amount and frequency of errors, a single transaction could be aborted. Or if the attack is considered more severe, all sensitive key data could be deleted from registers and (persistent) memory. For other types of tamper attacks, we can connect non-maskable interrupt lines to tamper-detection sensors that raise an interrupt when the enclosing of a device is opened or the temperature or voltage levels are outside the tolerated range. Finally, we can add a secure watchdog timer to protect against denial-of-service attacks by detecting that a task is taking longer than normal or by detecting that trusted software is not executed at regular intervals. In case of an expiring watchdog timer, the processor is reset and starts executing the initial, secure root-of-trust software again. On detecting a warm reboot due to the expired watchdog, this trusted software can take countermeasures like disabling a peripheral interface that overloads the device, or it can restore a trusted software image when the attack is caused by a recent insertion of malicious software.

There are a number of ways to protect against side-channel attacks at the processor level. Avoiding instruction timing variations due to data dependencies is an example. Multiplications by zero should take the same amount of time as non-zero multiplications, and taking a branch or not should not influence timing. Furthermore, designs should flatten the power profile of the processor as much as possible. However, since flattening of power profiles is not always fully possible, designers can implement in the processor hardware further countermeasures such as additional computations that randomize the power and timing behavior of operations. Adding this level of security increases computation cycles and consumes additional energy. However, we can minimize the overall energy consumption by applying the randomization to only the most critical sections of the software.

The final classes of countermeasures we'll discuss in this section thwart reverse engineering and IP theft attacks. Encryption and scrambling are the general techniques design teams traditionally use to protect application code and data. These techniques should be applied to memory, but we can also apply them to other elements of a processor. Figure 5 depicts an example where we integrate instruction encryption into the processor pipeline. Such an encrypted instruction processor pipeline provides a very high level of protection, since decrypted instructions are never stored in memory or in registers and are only decrypted just before they are used.

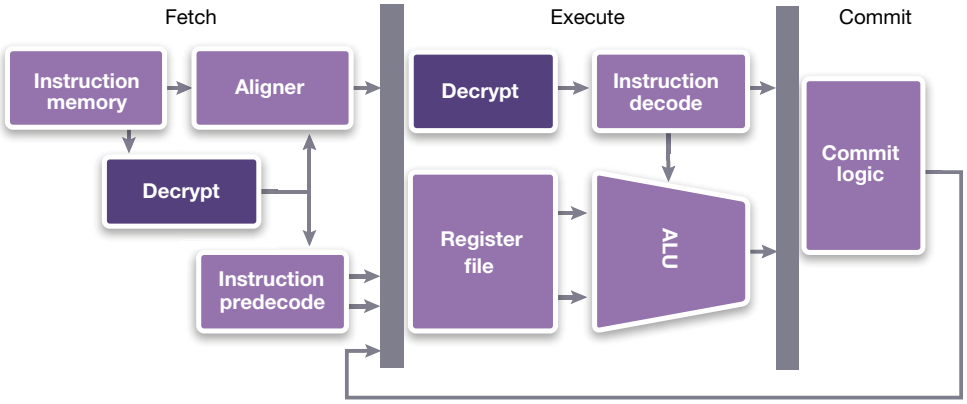


Figure 5. Pipeline encryption

Figure 5 shows how this can be implemented in a power-efficient three-stage pipeline by including decrypt logic in both the Fetch and the Execute stages. Since the decryption is executed in-line with the processor pipeline, the algorithms used in this case should be of limited complexity and logic depth.

Another element where encryption and scrambling can be used is on data memory. Since data can be both read and written, both encryption and decryption is required. Besides encrypting the data itself, it is also possible to rearrange the location in the memory where the data is stored by permuting the address lines. When we combine all of these techniques with the access control protection that is provided by an MPU, the techniques enforce each other and we can create a powerful, layered security solution.

Software-Level Protection Mechanisms

Now that we've gone over the system-level and processor-level protection mechanisms for edge node IoT applications, let's take a look at the many software technologies that complement the hardware mechanisms. We will be looking at software in a broader sense, including runtime software that is part of a final product and software tools we can employ to enhance the security of IoT devices.

The first software tools required for building secure IoT devices are tools for managing the device-specific and product-family-generic secrets, such as symmetric keys, device identifiers, and public key certificates. Although for simplicity developers often use a single key for protecting all devices from a product family, this practice implies that if the secret for one device is retrieved by an attacker, all devices of the family that share that key are immediately insecure. A better approach is to have device-unique keys, or a combination of device-unique keys and shared platform keys. These keys, device identifiers, and other secret data all need to be generated, managed, and provisioned to the devices during manufacturing in a secure way. Developers can use standard cryptography tools for key generation. Depending on the specific non-volatile key storage solution developers choose for an IoT product family — like secure NVM, fuses, or PUF — developers should use specific key injection tools in a secure way, typically by implementing this key provisioning in a trusted, secure environment.

"When designing IoT edge devices, architects must make key architectural decisions at the system, processor, and software levels."

For the memory encryption described in the previous section, as well as for firmware authentication and integrity checking, developers need tools that sign and/or encrypt firmware images. These build on the key management tooling for the authentication codes, certificates, and encryption keys, and can either operate as a post-processing step on binary images delivered by software build tools, or developers can integrate them into their software build environment. In the case of region-based encryption, the tools should apply different algorithms or use different keys for different parts of the binary images.

The last two types of tools prevent and detect potential security flaws in software, either at design time by performing source code analysis or during verification by performing automated security testing. There are many software quality tools that help improve general software quality. Fewer bugs mean fewer security risks. But there are also static analysis tools that have specialized support for finding the type of software errors that attackers typically exploit in different programming languages. The errors include buffer or stack overruns and null pointer de-references in C code, SQL injection, or cross-site scripting in web languages as JavaScript. An example of a tool that detects this is the Synopsys Coverity® Static Application Security Testing (SAST) tool suite.

Looking at security services provided by the platform software, two types that are specifically important for IoT systems that we must also consider are platform security and secure communication protocols.

Platform security features cover services like secure boot, access control, identification and authentication, firmware integrity assurance, runtime protection and application sandboxing, secure storage, secure in-the-field updates, secure provisioning, storage and management of keys, certificates and other data, device and feature activation, revocation and recovery, and personalization. Platform security starts with a hardware-guaranteed root of trust. The hardware guarantee typically consists of fixed boot ROM software that cannot be tampered with. Alternatively, we could also place the processor start code in reprogrammable non-volatile storage if the hardware supports a form of on-the-fly integrity and authenticity checking or instruction encryption that is strong enough for the specific IoT device and corresponding attack scenarios. In the latter's case, the processor hardware will detect tampering of the root-of-trust firmware. Starting from this trusted boot software, further platform infrastructure code as well as application code can be loaded from Flash or other background storage and then checked using the hardware and/or software cryptography functions of a device.

To secure communications between IoT devices, gateways, and cloud services, architects and their teams must make many tradeoffs when it comes to which communications security protocols to employ. As with cryptographic algorithms, it is highly recommended to use well-known, proven secure communication protocols, as they will likely already have gone through excessive vulnerability testing. Unfortunately, many of these existing protocols have been designed for more capable and more power-hungry devices than low-power IoT edge nodes and always-on wearable devices. A lot of research, development, and standardization are still ongoing in the area of lightweight but secure communication protocols for IoT applications.

Focusing on security, three layers in an IoT software stack correspond to different options for securing communication: the application level, the transport layer, or the network/data link layer. We can implement application-specific encryption, authentication, and/or integrity checks at the application level. Meanwhile, we can implement TLS and its datagram version (DTLS) on the transport layer. They provide confidentiality and integrity, and can provide both client and server authentication. The main benefit is that they enable secure communication over non-trusted network and transport layers. The main drawback is that they depend on public key cryptography and infrastructure that require more resources than solutions using only symmetric cryptography with pre-shared keys.

As an alternative, or in addition to the transport layer security, communication can be protected at the network or data link layer. Well-known examples are WiFi Protected Access (WPA) and IPSec. These provide similar protection as (D)TLS but at the network layer; thus all layers and applications on top benefit. This can, however, mean that communications by different applications over the same network interface are not protected from each other. Different variants of network and data link layer solutions exists, including efficient versions based on pre-shared keys that do not require the heavier public key algorithms and infrastructure.

When architecting an IoT system, design teams need to make complex, interdependent, and multi-disciplined tradeoffs. Security is an emerging system property that all teams creating IoT devices need to consider at all levels of their designs. Making the required tradeoffs is significantly easier by leveraging secure, proven building blocks that were designed with secure systems in mind and that were optimized for low footprint and energy. Depending on the specific IoT device threat model and the resulting security requirements, architects, hardware designers, and software designers can make tradeoffs and converge on the optimal mix, resulting in a secure architecture that will save your company hardware re-spin and software patch headaches down the road — thus allowing your company to maximize profitability over the long haul. ■

For a more detailed description of these tradeoffs, read the white paper, "Securing the Internet of Things: An Architect's Guide to Securing IoT Devices Using Hardware Rooted Processor Security."

About the Author

Ruud Derwig has 20+ years of experience with software and system architectures for embedded systems. Key areas of expertise include (real-time, multi-core) operating systems, media processing, component-based architectures, and security. He holds a master's degree in computing science and a professional doctorate in engineering. Ruud started his career at Philips Corporate Research, worked as a software technology competence manager at NXP Semiconductors, and is currently a software and systems architect at Synopsys.

"The trouble is, in the rush to connect everything through the internet... features, and reliability, sometimes security has been left out of the mix."

Software Is Everywhere — And So Are the Vulnerabilities

Robert Vamosi, Business Development Manager, Synopsys Software Integrity Group

Software is no longer limited to traditional computing platforms such as our personal PC or a corporate server. Almost every device today runs some software — from firmware at the chip level in our toasters to a complex operating system found within our smart TVs. Furthermore, life-critical products, such as automobiles, medical devices, and industrial control systems for the national critical infrastructure also depend on an increasing amount of software.

The first-generation internet, with its ever-growing number of individual websites offering all kinds of personal information sharing and transactions, is evolving into the "Internet of Things" (IoT), the combination of networked endpoint devices with complex cloud-hosted services connected through web-APIs. This provides new, unseen levels of functionality in a highly distributed manner. It also offers new challenges.

Smart What?

Consider the common radio. During much of the 20th century, the common radio consisted of circuitry you could purchase at Radio Shack for a few dollars and assemble yourself. Today, a digital radio is much more complicated; for one thing, there's software involved. Whereas you once used a simple variable capacitor to manually tune in specific AM and FM radio frequencies, radio signals today are digital, can be programmed in, and can also display on a tiny screen not only the song title, but the artist and maybe some advertisement as well. Sometimes known as HD Radio or DAB (digital audio broadcast), these new radios enlist various communications protocols to tune by station names rather than specific frequencies or access the current weather data for your local area.

Something similar has happened to TV. At one time a television was a box of tubes and wires with an antenna (makeshift or otherwise) that could pull in all over-the-air broadcast signals from the major television stations available in your area. Today TV signals are no longer analog; they're also digital. And they're no longer confined to broadcast frequencies; they're available over the internet. These new smart TVs allow you to access content other than just audio and video; now you can also access background information on an actor who happens to be on screen or a location where that scene was shot. Or purchase an item you see on screen over the internet without leaving the show you are watching.

IoT is changing the world. Whether it is chips in your car or in devices in your home or in devices on your body, the world today is running on software. Devices such as smart watches, however, do not have traditional full-blown operating systems. Instead they operate with a system on chip (SoC) and use a runtime operation system (RTOS).

There is a direct correlation between the addition of software and new features and more reliability. Smart circuitry is less likely to break. And smart tuning can be much more precise so the quality of sound and picture is high. The trouble is, in the rush to connect everything through the internet, to provide these new services, features, and reliability, sometimes security has been left out of the mix. And even though they lack a traditional operating system, the software—whether it is in the chips or in an app on your phone—can also be compromised.

Vulnerabilities found in chip-level firmware can be exploited because embedded software has not traditionally been designed for security and is difficult to remediate because the device itself lacks upgrade capabilities.

Apart from chip-level software there are myriad applications that interact with our smart devices. The apps on your phone, watch, or tablet that allow control of the TV, the refrigerator, the home heating and lighting systems. They rely on protocols such as Bluetooth and often enlist open-source software packages whose libraries may or may not be updated and therefore may or may not contain known vulnerabilities.

Given this and the fact that our smart devices collect and share sensitive data about the user or their environment, our software-driven devices are becoming an attractive new target for criminal attackers.

Jeep Hacking

Why is software security so important? With a radio or a TV, perhaps the worst that could happen would be someone hacks into your device and interferes with your ability to tune in a specific station, delays a purchase, or denies service entirely. You either get a new radio, pay another way, or find another form of entertainment. But in other cases software is life-critical and therefore not always feasible to replace.

Consider the Modern Car

Perhaps the best known example of a serious software vulnerability was last summer's successful remote access of a Jeep Cherokee. Journalist Andy Greenberg, from Wired.com, drove a Jeep Cherokee owned by security researcher Charlie Miller during rush hour on a highway in Missouri in mid-2015. Miller, along with Chris Valasek, demonstrated that they could remotely connect to the vehicle and take control. Greenberg said later he was terrified as the Cherokee lost acceleration and the ability to brake on a freeway on-ramp.

“Immediately my accelerator stopped working,” Greenberg wrote. “As I frantically pressed the pedal and watched the RPMs climb, the Jeep lost half its speed, then slowed to a crawl. This occurred just as I reached a long overpass, with no shoulder to offer an escape. The experiment had ceased to be fun.[1]

Chris Clark, Principal Security Engineer for Global Solutions at Synopsys, said: “What was compelling about the event, in my view, was not only was it a discussion, there was an actual visual to what was occurring and that visual was actually taking place on roadways.”

Photo: Andy Greenberg/WIRED.com

[1] https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/

“Indeed, we know now that former Vice President Dick Cheney had the communications features of his pacemaker disabled out of fear that someone might alter its signals….”

This particular “stunt hack” was not without controversy.

“The secondary to that,” said Clark, “is the manufacturer that got exposed in that particular event was aware of the vulnerabilities that existed and already had plans to release and patch these particular issues but they didn’t make it in time. That in and of itself was kind of an interesting revelation. As everything unfolded we found out, okay, this particular manufacturer understood the core issues but didn’t understand the criticality of it from a knowledge or research perspective. Because the event was already out there they had to switch to more of a reactive nature where they could have been a little bit more proactive early on.”

As a result, Fiat Chrysler America (FCA) immediately turned off all remote access capabilities through UConnect, a dashboard computer made by Harman, that had been compromised by Miller and Valasek. FCA also issued a patch for consumers to install on their affected vehicle models. The company, however, did not stop there.

The Jeep Cherokee hack did, to Clark's point, drive the type of urgency within the auto industry that the mere publication of yet another academic paper on the subject would not. Within days, representatives of several major car manufacturers, led by FCA, began discussing ways they could get ahead of the security issues that might already be in vehicles on the road today, and avoid those issues in cars still on the drawing board.

At the time, in August 2015, these automotive representatives happened to be attending the annual security conference in Las Vegas, Nevada, known as Black Hat USA. During that conference, Synopsys had several conversations with the major auto manufacturers and their suppliers. Those conversations could have ended there. Instead, they continued throughout the fall.

Six months after the Cherokee hack, automotive representatives conferred once again on a cold Friday morning just outside of Detroit, Michigan, with a few security companies (including Synopsys) in attendance, agreeing in principle to form a software security working group. Initially named “Featherstone” after the street address where the initial meeting was held, it is now a functional working group within the Society of Automotive Engineers (SAE) with individual members from a variety of associated automotive OEMs, suppliers, and others actively working to define best practices for secure automotive software development.

Software Is Getting Personal

Another life-critical area where software plays a major role these days is medical devices, whether within a hospital environment or within a human being. In early 2008, the software used in medical devices such as pacemakers was shown to be vulnerable to various forms of attack. Indeed, we know now that former Vice President Dick Cheney had the communications features of his pacemaker disabled out of fear that someone might alter its signals (and there was a fictional account of this in the popular television show Heartland).

There are, however, a couple of compelling contrasts between the automotive industry response and the medical industry response. Whereas the medical community has had safety security practices in place and a number of years to absorb the vulnerabilities found in their devices, these best practices aren't always as prominent today as they should be. And while the automotive industry also had safety and security practices in place, it actually stood to benefit from the most from the recent disclosures.

“In that respect,” said Clark, “even though connectivity happened at a later time from a vehicle perspective, it seems that [the automotive OEMs] looked at what has happened in other industries, other verticals in history, and took it seriously. They started making those implementations early on. And because of the types of vulnerabilities we see out there, we can see that because it was an activity that started early on in the manufacturing and development process, that their security practices are much richer and better implemented than what we see in medical practices in medical device manufacturers today.”

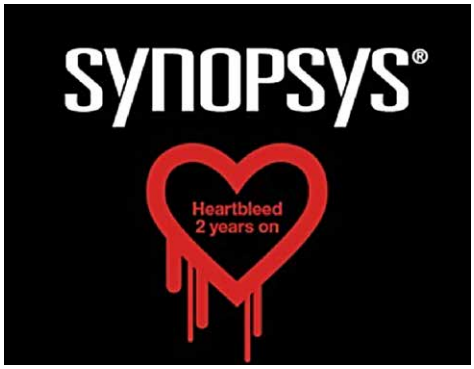
Trust but Verify (Often)

Mike Ahmadi, director of Critical Systems Security at Synopsys, says that supply chains need to be tested repeatedly. “For example,” he said, “despite the fact that we have a lot of best practices that we do for air travel, every single time someone flies a plane, [the pilot] goes through step by step and verifies and checks off everything on a known checklist. This happens regardless of how much they trust the organization that builds and maintains the airplane. And, by the way, they find things quite frequently. I know this because I’ve had plenty of flights delayed for that reason.”

The reason the checklist is necessary, says Ahmadi, is that critical systems are not static—at least the safety and security aspects are not. “The problem with security is [it's] dynamic," he says. "You can verify something today — and they could be good for a while — but a year from now…the entire vulnerability and attack surface has completely changed. You may be able to (depending on the criticality for a low-criticality system) say, 'okay, I can shorten the mean time between verification,' but for high-criticality systems you should be verifying really as often as possible.”

This is true with software in the cyber supply chain. You can add code from a common open-source project, but have the libraries been updated? Or worse, are there new vulnerabilities possible against the existing project?

In 2014, researchers at Synopsys independently co-discovered a severe vulnerability in OpenSSL that immediately put more than 600 million IP addresses at risk to compromise overnight. The flaw in the implementation of the heartbeat feature of SSL had been present in the product for nearly two years. Without continual testing, without going through step by step and verifying the individual components in a given software package, it is impossible to assess your organization's risk today or in the future.



That's where Synopsys offers value.

“We provide automated testing tools,” said Ahmadi. “You could do all of this manually. You can manually go through line by line of code. You can manually look up CVEs and CVE scores and CWEs to define your metrics. You can manually comb through and manually create test cases that could fuzz something at a protocol level, okay—or you could connect them to our automated testing tools, push the button, and wait.”

Whether it's a dashboard component of a car or a device implanted somewhere in your body, the software running that device is just as important—if not more—than the software you have on your desktop, laptop, or mobile device. If you manufacture one of these devices and you don't currently test for software quality and security, you should. And if you fix these vulnerabilities early in the software development lifecycle, then it will save you a lot in remediation and, at a minimum, brand reputation.

Ahmadi added, “I won’t say the software is now error-free but you’ve dramatically reduced the number of errors when you automate the test tools, and you can get more done in a shorter period of time.” ■

About the Author

Robert Vamosi, business development manager in Synopsys’ Software Integrity Group, is the author of *When Gadgets Betray Us: The Dark Side of Our Infatuation with New Technologies.* He is also featured in the history-of-hacking documentary film Code 2600. For more than fifteen years Vamosi has written about information security for such publications as *Forbes, CBS News, and PCWorld*.

Agile Development for Application Security Managers

Software Integrity Group Staff

In today's competitive business environment, it is more important than ever to develop applications not only accurately but quickly. The traditional "waterfall" method is effective, but requires so many steps that the process cannot keep up with today's software development needs. Agile is a development methodology that speeds up software development dramatically, along with several other benefits that make it a popular methodology.

Vulnerabilities in applications pose an ongoing threat to business-critical data more than ever before. Organizations are faced with ongoing, persistent threats that originate in their web applications. Many think that agile software development and application security cannot co-exist; in other words, that application security is a requirement that agile development teams cannot meet. Agile development is just too nimble and lean—it cannot be bothered with security—and any attempt to introduce application security into the process will have a great negative impact on the development process.

Having said that, organizations do report success with implementation of application security within the agile development process. How can this be achieved? Let's analyze agile development from the standpoint of application security, and look at ways potentially to effectively implement security into the agile development methodology.

Agile Principles Overview

Pure agile development is defined in "the Agile Manifesto." While most organizations do not adopt the pure agile form, many implement a methodology that relies on agile principles. In short, they use agile as a framework. Here is a summary of the agile principles many development teams tend to adopt.

Responsibility at the hands of the developers: Development teams are given responsibility and are trusted to get the mission accomplished. The best method of communication is frontal communication among all stakeholders.

Code is constantly uploaded and updated: This allows changes to occur on the fly, allowing the end-product to adapt as needed. The lifecycle of an agile project consists of nearly constant development and testing, rather than distinct states.

This also incorporates the extreme programming (XP) software engineering practice of merging all working copies with the mainline source up to several times per day. This prevents long-term integration problems and as a result the current status of the project is always changing.

Requirements arrive late in the process. Agile teams know that requirements will change and evolve through the process, meaning that early investment in documenting requirements is wasted. Early on, there is just enough envisioning of requirements to identify the scope of the project. Firm requirements will be determined deep into the process when the entire team understands better what the end results may be.

Customer-oriented requirements: Projects are completed with customer collaboration, but the customer ultimately determines the requirements and scope of the project. The ultimate goal is to deliver working software to the customer in a timely manner.

User stories: A user story is a short, simply-written statement that explains what a user needs to do as part of the job function of the project. This is one of the facilitators of requirements management.

Automated, ongoing testing: Automated testing and test cases allow more frequent testing throughout every step of the agile development process. This allows for quality software even within the shorter development cycles that the agile method produces.

Lean development: The agile methodology uses the principles of lean manufacturing in the software development process. This minimizes waste, empowers the development team, and delivers the finished project as quickly and affordably as possible.

Business people work together with developers: Daily, ongoing cooperation between developers and stakeholders is the only way to achieve the goals of agile development.

Robust, working software at a constant pace: The key measurement of delivery is working software that does what it should. This should be delivered at a constant pace. The focus is on delivering excellent software.

A process of self-improvement: Periodically, the teams should stop and reflect among themselves on problems in the current process and how to resolve these problems.

Achieving Application Security in Agile

In a nutshell, software developers use agile to provide quick development guidelines that require less need for discussion and requirement planning. Development teams use this method to satisfy customer needs within a shorter period of time. Customer feedback and testing put the product through a number of iteration cycles or sprints until eventually the application meets the goals decided by customer requirements and feedback.

So, you may wonder how secure development and application security testing fits into this process. Waiting until the end of the process can expose or even cause problems that essentially put the project back to square one. So it is important to work security into every step of the process. The following principles show how application security can be achieved, even during fast-paced development that use the agile methodology.

Because development through agile is a team effort, application security must be a team effort too. Agile is known for its leanness and speed—and to implement application security in the process, the same principles ought to be used.

Define Clear Requirements

Development and quality assurance teams are often baffled by requirements presented to them by security. In many cases, developers feel that security procedures are redundant and exist simply to generate more work for the development team, especially when application security testing does not take place until the very end of the project.

Because of this, it is important to define clear expectations, ideally from the very beginning. The development and testing teams need to understand what is the desired level of security and the meaning of achieving this level, as well as which security tests will be conducted and what results are expected.

"When it comes to security… you must make certain that the tools you use facilitate interactions between team members by providing a common language understandable by everybody, and that they do not encumber the process."

Giving a better description of what the team should focus on in terms of security, information about the testing process itself and the logic behind it can help the developers work security into the project as a whole. Answering the following questions for the developers is a good first step in defining the requirements:

- What are the specific areas of focus in developing securely and testing for security?
- Do these tests replace periodic penetration tests and security audits, or are they utilized alongside these testing methods?
- How often should developers test for security and who on the team is responsible for doing these tests?
- What security standards should the development team strive to meet or exceed? (this could be industry standards like OWASP, PCI-DSS, internal organization requirements or something else)

After answering these questions, the development team can best integrate application security into the agile development process from a place of understanding and cooperation, not from a place of suspicion and ambiguity.

It is important to provide not only requirements but also support toward the goal of achieving them, whether by training, secure frameworks to work with, etc.

Application Security As a Natural Part of the SDLC — Seeker

Synopsys' Seeker® tool is the runtime code and data analysis application security testing solution for the software development lifecycle. By analyzing application behavior in response to simulated attacks, the Seeker tool detects code vulnerabilities that pose a real threat. It assists in vulnerability management by generating exploits that demonstrate the risk to business-critical data. The Seeker tool is the perfect application security testing solution for the SDLC; it can be fully automated and works seamlessly in agile and continuous integration environments.

The Seeker solution includes the following benefits:

- It integrates application testing seamlessly into the development process and with functional testing.
- Build servers, and any other existing automation via a powerful out-of-the-box interface. You just put Seeker where you put all the rest of your automatic testing.
- Vulnerable code is highlighted and remediation provided, resulting in minimal work for developers and testers.
- Risks are explained in simple terms, allowing stakeholders to easily prioritize and build a vulnerability management plan,
- Both the cause and the fix are shown, allowing developers to recreate the scenario, see the code that is causing the problem, and get an easy remediation solution—all in one place.
- Vulnerabilities are managed as any other bugs.
- Speedy testing processes allow full testing of the application in a short time, for new code or as part of regression testing.

Using the Seeker solution, an agile development team can implement security during the development process. This offers a huge return on investment as well as improved customer satisfaction, since projects are completed on time and within the constraints of the agile framework. The Seeker tool is simple and delivers nearly immediate results with little effort or change in the development process.

Work with the Process, Not Against It

Agile development teams have certain processes that they employ during the project lifecycle. Like nearly everything involved with agile, these processes are meant to be lightweight and lean. The best way to successfully integrate application security into agile development is to work alongside existing interactions. This way the created code meets all the requirements of the project, but it can also be secure and of high quality. "The Agile Manifesto" states: "individuals and interactions over processes and tools." When it comes to security, you cannot completely forget tools, but you must make certain that the tools you use facilitate interactions between team members by providing a common language understandable by everybody, and that they do not encumber the process.

This means, for example, if a team is using specific bug tracking software, then security issues should also be delivered as bugs using the same interfaces. If there is existing automation and regression testing, choose the application security tools that can be integrated into this process. Prefer tools that deliver results in a language that developers understand—with location of the vulnerability identified in the code and preferably with clear explanations and demonstration of vulnerability risk level to avoid arguments over priority of fixes.

Essentially, instead of making security a separate process that can lead to end-of-project setbacks, security must be integrated into every step of the agile process.

Accommodate Frequent Code Changes

Agile not only involves frequent code changes, it encourages them.

As such, it is very important that security testing caters to these same standards. Application security within the agile methodology must also change just as quickly. Application security that provides this need of ongoing security will provide protection without complicating matters for the designers, leading to huge hurdles or forcing these developers to work outside of agile principles.

This means tools that take a long time to run are not effective in these environments, nor are tools that require manual interpretation of results. This is due to the likelihood that by the time the tool is done testing and a reviewer from the security team is done reviewing, the new code would already be in production for days or even weeks.

Create Security Stories

Nearly all teams work with at least some level of delivering requirements as user stories. Presenting application security in the form of user stories keeps the flow familiar and works within the standards of an agile project. Additionally, one of the most important things to remember is that security is part of everyone's job, not something that is entirely pawned off onto one person or team. By formulating security as a user story, it reminds the developers of this important part of their task, and presents security as just another aspect of the development and testing cycle.

Help Create an Agile Application Security Workflow

Explain to agile developers what is expected in terms of security. Then work directly with them to create a workflow that fits in with current habits, iterations, and deadlines. Questions often asked by development teams are:

- Who should run security testing; should each developer run on their own code, or maybe have one QA member who is responsible for security testing?
- How often should security tests be performed; should they be on every piece of code or after integration?
- Who should the results be delivered to: development or security?
- Who is responsible for signoff?

The meaning of an agile application security workflow is creating a development process with embedded application security throughout all its phases, while still keeping the agile principles of being lean and quick.

Provide a Training Program

Many developers do not have the training necessary to properly understand application security and perform the testing. Even if a training program is in place, developer turnover makes it very difficult to have everybody always up to date. Before you hand off responsibility to a development team, you should provide enough information to make the process easier.

Use application security tools that involve training in the process. This training will pay off not only for the project at hand, but also with future projects developed by a similar method. This is one of the benefits of agile development—future projects are even easier.

Remember that training does not have to mean two weeks of training covering a multitude of topics—some relevant, some not—followed by an examination and usually resulting in the developer instantly forgetting most of what they learned. Training can mean, for example, that if a developer has a specific vulnerability in their code, they will need to do a short training on this vulnerability, or training on specific vulnerabilities related to the application they are working on.

Don't be Afraid to Make Mistakes and Improve as You Go

Agile development is all about learning as the project proceeds. Any iteration includes improvements and changes, eventually moving toward the end result. The application security process is completed the same way, with developers making changes and improvements along the way.

Secure software, developed by any means, comes from properly testing and following proper security measures. The common perception that agile development methods cannot embrace secure coding practices and application security testing is, for the most part, false. With some flexibility, it is possible to integrate application security within your agile development system.

Developers should pay attention to the risks of not incorporating security within agile development. When software is not properly tested for security, there is a risk of developing insecure software that can lead to data loss and programs susceptible to hackers. While there is a cost to security testing, the cost that can occur due to improper testing will generally dwarf this cost. ■