SYSTEM VALIDATION: AN IN-DEPTH COMPARISON OF FPGA PROTOTYPING TO EMULATION AT SCIWORX

VOLKER WEGNER, CADENCE VERIFICATION ACCELERATION
CADENCE DESIGN SYSTEMS

With the worldwide trend toward digitalization, design and software complexities are steadily growing while design and product lifecycles are shrinking. Hardware, software, and overall system components all must be ready at the same time. And system functionality must be verified in conjunction—prior to fabricating silicon. As a result, design and software teams must improve productivity and quality.

Paradoxically, verification is a major roadblock in creating the final product and a key component of timely market release and success. Prior to real silicon there are only three verification options: FPGA prototyping, simulation, and emulation. This article summarizes the Sciworx verification team’s experience with a 1.2 million-gate multimedia design IP core including an ARM9 family processor. The Cadence® Incisive® Enterprise Palladium® system was evaluated against FPGA prototyping.

Sciworx provides various synthesizable IP blocks and subsystems, such as video encoders and decoders, for the multimedia space. Especially in the consumer electronics market, product lifecycles shrink as feature sets and resolutions steadily grow, resulting in higher demands on project schedules and product quality. Usually the verification tasks for block, chip, and system levels represent the biggest portion of the entire project effort, which includes testing of firmware and even application software—ideally before silicon is available.

Sciworx verification concepts were all optimized for speed. SystemC® models, along with a generic FPGA prototyping board, formed the basis for fast and efficient hardware/software co-verification. But as customers are demanding more and more streams to be verified before RTL delivery, a gap in the existing verification flow became apparent.

All of these factors called for a revision of the existing Sciworx verification flow. Sciworx and Cadence jointly conducted two verification assessments to analyze the existing Sciworx verification flow and explore options for improvement. After identifying significant performance hurdles and productivity limitations, Sciworx chose emulation to replace the existing FPGA prototyping verification strategy for their most important projects. Emulation proves to fill the verification gap by shrinking design cycles, improving productivity, eliminating risk, and improving overall product quality.

IDENTIFYING THE VERIFICATION CHALLENGE

Video codecs provide compression/decompression of video material for storage on HDD (hard disk drive) and DVD (compression), and for video playout of compressed movies (decompression). The need to support a variety of formats results in exploding verification complexities. Each design needs to be verified against hundreds of test video streams, each one consisting of hundreds of frames (pictures). Various compression standards and display standards need to be supported as well.
Verification of a previous design at Sciworx required processing 15,000 NTSC frames. It took three days to complete this task using the FPGA prototyping methodology. The next project faced a regression suite three times as big; missing the schedule would put the whole project at risk. Future verification requirements at Sciworx were expected to grow beyond 500,000 frames. And growing software content would need to be verified as well.

All of these factors called for an assessment of the existing verification environment used at Sciworx.

**ASSESSING THE VERIFICATION CHALLENGE**

IP verification environments require the modeling of complete systems, including additional components around the IP. The Sciworx verification methodology was built around performance optimization: SystemC simulation models, instructions set simulators (ISS), and bus functional models (BFM) for simulation complemented by a Sciworx FPGA prototyping board called SciBoard.

The SciBoard was designed as a regression and prototyping environment for all Sciworx design projects. It is based on three Xilinx Virtex 8000 FPGAs. Project-specific components can be added to the system board through general purpose ports. A software debugger can be attached, allowing firmware and application software to run while validating the design under test.

Verification of a DivX ASP SD encoder (the DXVE) required an ARM9 family processor (connected to the debugger in the ARM RealView Developer Suite), an AHB arbiter, and an SDRAM as additional components connected to the debugger in the ARM RealView Developer Suite. In simulation, they are represented as SC, ISS, and BFMs, which can access memory contents on their own. On the SciBoard, they are mapped to FPGAs and extra hardware. To access data, an Ethernet interface is added. An additional AHB port is used to mimic bus load:

The Sciworx verification team’s experience with the FPGA prototyping-based verification environment showed a number of limitations.

**Limited debugging:** Debugging capabilities within FPGAs are limited, especially when they reach full capacity. The more FPGA logic that is used for representing design content, the less logic is available for on-chip probing. Changing probes or trigger definitions is meant to be quick with incremental compilation, but as devices fill, the success rate of incremental compilation decreases.

**Long compile times:** FPGA compile times are in the range of four hours or more. Once capacity boundaries are approached, compile times increase exponentially. Consequently, RTL changes can be implemented within an FPGA only once or twice per day.

**Slow bring-up time:** FPGA board bring-up exposes timing- and board-related problems, distracting from the intended verification goals. These problems cannot be reproduced in simulation and therefore must be debugged on the prototype. The worst-case scenario: when capacity or interconnect can no longer fit and repartitioning becomes necessary. This is equivalent to starting half-way from scratch. Alternatively, it is possible to leave out less important blocks. The result is growing differences between simulation, the prototype, and final silicon. The combination of all these issues regularly results in excessive resource allocation and multiple months of bring-up time.

**Increased risk:** Once the prototype is functional, it neither allows for easy debugging nor for quick bug-fix testing. This is problematic in general but is especially painstaking for the last handful of bugs, which appear after long runtimes. Debugging those remaining bugs sometimes becomes a nightmare and, in worst cases, adds weeks to the overall project time.

**Greater differences among environments:** Complicating things even more are the differences in how memory is accessed, differences in modeling for different cores like the processor, as well as the prototype’s layout representing a different top level than what’s being simulated. For performance tuning or simply to keep up DRAM refresh rates, even the clocking scheme might differ.

**Raw performance versus overall throughput:** Finally, the overall throughput of the SciBoard turned out to be much less than expected from the raw clock-speed numbers. Although the clock speed was around 1/5 of the final design, the overall image data throughput turned out to be just 1/500. The reason is the way memory contents are transferred to and from the SciBoard.

*Figure 1: Existing verification environment*
ADDRESSING THE VERIFICATION CHALLENGE

To increase productivity and reduce overall risk, the Sciworx DXVE verification team needed to achieve as many improvements as possible. They wanted short turnaround times and a comprehensive debug environment to support simultaneous HW/SW debugging. Overall, data throughput needed to be increased and, since the schedule was already tight, impact on the team needed to be minimal.

Sciworx decided to evaluate an emulation approach in parallel to the existing verification flow. As a bonus result of this assessment, the verification bottlenecks of the existing SciBoard improved substantially: compile time went from half a day down to four hours, and overall data throughput went up 10 times from 0.1 frame/sec to 1 frame/sec.

The emulation setup was implemented as close to the SciBoard setup as possible, although inherent limitations were known upfront. The goal was to start as smoothly as possible with minimum impact on the team. This allowed for debugging the related issues of the SciBoard. The downside was reintroducing the inherent performance bottleneck of using the Ethernet interface for data transfers. That way the emulation setup wouldn’t reach the overall image data throughput of the SciBoard, but this was acceptable since both platforms would be available for the project. If successful, the data transfer mechanism would be changed later to directly accessing the memories inside the emulator (path B).

Most of the entire SciBoard was mapped into the emulator. Together with the design, several IP blocks went into the emulator: ARM9 family processor, DRAM, AHB arbiter, and all embedded memories. For speed reasons, the Ethernet interface was placed outside the emulator and was used to load/dump image data to and from memory (path A). A debugger from the ARM RealView Developer Suite was connected to the environment.

EXPERIENCING EMULATION

Sciworx’s past experiences with Quickturn emulators of the late 1990s exposed problems with functionality and turnaround times as a result of FPGA-based systems. Functionality variations resulted in frustration; memory modeling was only for experts. The emulation runtime performance was determined by trial and error and varied over compiles. Typical compilation time was overnight.

Today, all such problems are history. The Incisive Enterprise Palladium system demonstrated very high quality and addressed all aspects of interest:

**Fast and easy RTL compilation:** Memory modeling is as simple as writing a few lines of RTL; even behavioral memories are accepted. Once defined, the target interface timing is easy to define and fix. The emulation runtime performance is simply reported. The compile time from RTL into the box for 1.2 million gates is ten minutes on Sun and five minutes on Linux, compared to four hours for the SciBoard. This enables verifying a bug fix within a coffee break.

**Simple flow integration:** During setup of the environment, the emulation-related compile flow was integrated into the Sciworx design flow. The associated effort took less than one week and the integration was easy. Once the emulation environment was handed over, the verification team only needed to know three commands: prepare (for library preparation), compile (to map the RTL to the Palladium system), and emulate (to start emulation). As a result of this learning process, the next Sciworx project—Muvistar, a mobile companion chip with multimedia capabilities—went into emulation within one day.

![Emulation environment](image.png)

*Figure 2: Emulation environment*
Remote access with no downtime: After the emulator was installed with the in-circuit verification environment in Hannover, Germany, almost no additional local access was required. With the Palladium emulator’s remote access and multi-user capabilities, the verification team (located 100km away in Hamburg) was able to use the system almost as if it were next to their desk. Within six months of operation there was virtually no emulator downtime.

Fast RTL debug: RTL-based debugging is similar to simulation and much faster than expected. The Palladium system’s inherent simulation-like features cover a debugging GUI, full design visibility, instant trigger changes, easy memory access, and force/release of any design signal. The most difficult bug to find was one causing the design to hang up after about one hour of in-circuit emulation. This is equivalent to half a trillion cycles. The emulator’s logic analyzer was simply triggered by inactivity of certain signals. Without emulation, this problem would have easily caused the schedule to slip several weeks.

Efficient software debug: The biggest surprise was the software debugging performance. Because 1MHz is a relatively low emulation speed, the expectation was low but turned out much better than expected and is fast enough for effective HW/SW co-debugging. Software debugging means simply to use the debugger in the RealView Developer Suite with its inherent capabilities, such as register and memory access (path C), single stepping, and breakpoints.

True system-level HW/SW co-verification: Palladium systems support dynamic targets by running the clocks continuously at fixed frequencies. This enables direct connecting with the debugger in the RealView Developer Suite, enabling both views (the designer's RTL view as well as the software developer’s view) at the same time. The resulting environment allows for true system-level HW/SW co-design and co-debugging.

Script-based runtime control: Because all runtime features are available through Qel—a tcl-based control language—the Palladium system allows easy integration into runtime environments like regression testing or automatic system configuration (including, for example, memory initialization or performing a board reset).

Easy setup: Because an emulator bridges the gap between hardware prototyping and software simulation, it allows the environment to stay much closer to what is simulated. Compared to prototyping, the effort to create and maintain different views is substantially decreased, allowing engineers to concentrate on the main verification tasks.

With the Palladium emulator, the DXVE verification team now benefited from a much more effective debugging environment and could eliminate substantial risks from the project. As a result, the strict time schedule was met and emulation paid off very well.

**COMPARING COST, EFFORT, VALUES, AND RISKS**

After six months of hands-on experience with the Palladium system, the time came to compare the cost, effort, values, and risks of FPGA prototyping versus emulation. H4VD, a H.264 HP HD decoder and the most critical future Sciworx project, was selected to do an in-depth comparison of the two possible scenarios.

Initial discussions with the H4VD management team brought up interesting points. Most engineering time is spent debugging rather than running the regression. Turnaround times of two or three days for a regression suite of more than 500,000 NTSC frames would be acceptable. Since H4VD could not be mapped on the given FPGA boards, either a new one needed to be created or one needed to be purchased from an FPGA-prototyping vendor. With an additional effort of one work week, the two options went into an in-depth comparison.

While overall cost is fairly close, the effort required for an emulation setup is much less; predictability of the results is higher; and, the overall risk is lower. FPGAs look inexpensive at a first glance. However, based on past experience, cost for design, creation, and deployment add up and are much higher than originally thought.

With direct access to the emulator’s memories and multi-user capabilities, the overall throughput of an emulation environment turns out to be on the same order as FPGAs. But looking at regression is only half the picture.

Given that most time is spent in debugging, the overall speed and efficiency weighs a lot in the Palladium system’s favor. RTL debugging and HW/SW co-verification greatly increase productivity. Additionally, emulation offers flexibility in terms of multi-user, multi-mode, and remote access, allowing different projects and locations to share the emulator simultaneously, even in different modes of operation.

The ability to fully control the emulator from a UNIX environment means much less effort on creating the necessary environment. With almost no testbench it is possible to pump the complete regression suite through the design in the emulator. In contrast, complex scripting is necessary to achieve fully automatic regression testing in an FPGA prototyping solution. This is one major drawback of FPGA-based testing.
The results are compiled in the following table:

<table>
<thead>
<tr>
<th>Thread/Opportunity</th>
<th>FPGA prototyping</th>
<th>Palladium emulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>System cost</td>
<td>$100K</td>
<td>$240K</td>
</tr>
<tr>
<td>System performance</td>
<td>8MHz/20-50MHz</td>
<td>800KHz</td>
</tr>
<tr>
<td>Complexity of validation environment</td>
<td>Very high (5-8000h)</td>
<td>Medium (500h)</td>
</tr>
<tr>
<td>Required adaptation of verification environment</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Effort required to develop additional software</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td>SW debugger support</td>
<td>Very good</td>
<td>Good (speed)</td>
</tr>
<tr>
<td>HW debugger support</td>
<td>Poor</td>
<td>Very good</td>
</tr>
<tr>
<td>HW/SW co-debugging</td>
<td>Poor</td>
<td>Good</td>
</tr>
<tr>
<td>Turnaround time on HW changes</td>
<td>4 hours, 4 CPUs</td>
<td>5-10 minutes, 1 CPU</td>
</tr>
<tr>
<td>Turnaround time on SW changes</td>
<td>Unknown</td>
<td>Seconds</td>
</tr>
<tr>
<td>Software upload via script</td>
<td>Maybe</td>
<td>Yes</td>
</tr>
<tr>
<td>Risk of trouble on architectural changes</td>
<td>High (pins, capacity)</td>
<td>None</td>
</tr>
<tr>
<td>Remote access</td>
<td>Maybe</td>
<td>Yes</td>
</tr>
<tr>
<td>Unix-based scripting for automated tests</td>
<td>Maybe</td>
<td>Yes</td>
</tr>
<tr>
<td>Requires local host</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>System requires full-time person to support</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>System usable by other projects on low load,</td>
<td>Requires changes</td>
<td>Yes</td>
</tr>
<tr>
<td>multi-user/multi-mode support</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Risk of show stopper</td>
<td>Medium</td>
<td>Low</td>
</tr>
</tbody>
</table>

*Figure 3: Comparison of FPGA prototyping to emulation*

Another interesting point: During the FPGA compile time, emulation verified more than 5,000 of the most complex frames. The reason is the fast compile time of five minutes from RTL into the box. Given that most bugs show up within the first few tests, emulation provides much higher productivity.

**CONCLUSION**

Sciworx used emulation in the late 1990s and FPGA prototyping during the last five years. A decision between using FPGA prototyping or emulation is dependent on several factors. While cost of ownership is in the same range for both options, Sciworx determined the value of emulation indispensable, especially in the area of debugging. The decision to introduce emulation in general was based on a relatively small design of 1.2 million gates including an ARM9 family processor. Growing design sizes and shrinking time schedules supplement this decision even more.

At first glance, emulation looks expensive, but it provides great flexibility and automates setup and maintenance of the verification environment. Significantly fewer human resources are necessary, while debugging productivity increases dramatically and predictability of results is very high.