### Virtual Roundtable: User Perspectives

Several high-level-synthesis users, whose experience spans the range of commercially available HLS tools, were recently invited to a virtual roundtable to share their HLS experiences. The various questions provide context for how they have used HLS, the benefits they have derived from it, and areas for improvement that they would like to see in the future.

<table>
<thead>
<tr>
<th>Participant</th>
<th>HLS Tool Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ashish Trambadia</td>
<td>Cyber Work Bench (CWB) from NEC</td>
</tr>
<tr>
<td>Gael Clave</td>
<td>Esterel Studio from Esterel EDA Technologies and/or PICO Extreme from Synfora</td>
</tr>
<tr>
<td>Hiroyasu Hasegawa</td>
<td>Cynthesizer from Forte Design Systems</td>
</tr>
<tr>
<td>Nitin Chawla</td>
<td>Catapult C from Mentor Graphics</td>
</tr>
<tr>
<td>Suhas Pai</td>
<td>Bluespec SystemVerilog (BSV) for hardware implementation and SystemC for testbenches</td>
</tr>
<tr>
<td>Gernot Koch</td>
<td>Cadence C-to-Silicon. We've also been using Synopsys SystemC Compiler for a long time, but only in a niche.</td>
</tr>
</tbody>
</table>

**D&T:** Which HLS tool are you predominantly using in your design flow?

- **Trambadia:** We use Cyber Work Bench (CWB) from NEC.
- **Clave:** For production devices, we use both Esterel Studio from Esterel EDA Technologies and/or PICO Extreme from Synfora.
- **Hasegawa:** Cynthesizer from Forte Design Systems.
- **Chawla:** We use Catapult C from Mentor Graphics.
- **Pai:** We use Bluespec SystemVerilog (BSV) for hardware implementation and SystemC for testbenches.
- **Koch:** That would be Cadence C-to-Silicon. We've also been using Synopsys SystemC Compiler for a long time, but only in a niche.

**D&T:** What is your primary target technology?

- **Trambadia:** Most of our designs are targeted primarily for 130-nm and 90-nm ASICs. Some designs, although very few, are targeted to Xilinx FPGAs.
- **Clave:** Our target technologies are ASIC and FPGA for prototyping.
- **Hasegawa:** Deep-submicron ASICs and FPGAs.
- **Chawla:** Nanometer ASIC technologies.
- **Pai:** In our case, the target technologies are 4G wireless platforms and chips.
- **Koch:** Various ASIC technologies from 0.5 micron down to 90 nm. For internal purposes, we are also using FPGAs from Xilinx.

**D&T:** What are the typical applications for which you design with HLS and why?

- **Trambadia:** In the past few years we have used HLS tools for designing SoCs and different types of IP blocks. These include AMBA [Advanced Microcontroller Bus Architecture] bus IP blocks like AXI [Advanced eXtensible Interface], AHB [Advanced High-performance Bus], APB [Advanced Peripheral Bus], AHB-Multilayer, and AMBA cross-bus bridges for AXI, AHB, and APB. Similarly, HLS tools have also been used for cryptographic IP blocks like ECC, RSA, and Viterbi. A few image-processing IP blocks and computational IP, like CORDIC, were other candidates that were developed using HLS tools.
- **Clave:** Our target technologies are ASIC and FPGA for prototyping.

For bus IP modules, it is necessary to have reconfigurability so that the user can easily create a bus with a different number of master and slave interfaces and different data bus widths, which is very difficult to do in RTL. In such a scenario, we find using HLS tools very convenient as we can easily regenerate.
RTL with optimized areas and timings from the behavioral source.

For algorithmic IP modules, earlier when we designed in RTL we faced many issues to achieve the desired latency and area, while with HLS usage we have been able to achieve very good area and timing results.

With the increasing complexity of algorithms and time-to-market pressures, it becomes challenging for RTL developers to achieve high performance in area and timing. Consequently, with the help of HLS tools we have been able to reduce development times by 40% to 60%.

**Clave:** We use HLS tools for designing multimedia codecs, modems, encryption algorithms, or for designing hardware infrastructure such as DMA, cache, memory management units, and memory controllers.

**Hasegawa:** We've used the HLS tools for copy, printer, digital camera, and 3D graphics chips. The image processing applications we have are computationally intensive with few interrupts.

**Chawla:** Our applications include complex wireless and multimedia application engines. These complex systems are often have high-performance and aggressive quality of result [QOR] needs. The time to market to design these application engines (from developing the algorithms to implementation in silicon) is very short. HLS methodologies enable high-level IP reuse and design space exploration to build these application engines with good QOR for the target ASIC technology thus providing high levels of design productivity.

**Pai:** Synthesizable transactors is the application we use HLS tools on. The reason we use HLS tools is primarily because of the productivity gain that comes from ease of writing transactors and being able to quickly configure them.

**Koch:** We are using HLS primarily on our algorithmic video blocks. Our system engineers are writing software models of the blocks, and HLS tools enable us to use those software models with minor modifications as the starting point for synthesis, down to the gate level.

**D&T:** What has changed in recent years enabling HLS to become successful? Do you think the available HLS tools fit well with your design flow?

**Trambadia:** Rapidly changing product requirements with more complex designs are enabling HLS to become successful. HLS tools are best suited for developers to write algorithms based on new design requirements. It is easier to regenerate new RTL designs instead of modifying existing RTL designs. Apart from optimized RTL generation, HLS tools also provide an extra mechanism for verifying design functionality with behavior simulation and for generating an RTL testbench for design.

**Clave:** What has changed is the maturity of both tools and people skills. Overall, yes, these tools are well integrated and are part of our design flow, but the integration must be planned. Tools interoperability is essential: for example, interoperability between Esterel and PICO, and between PICO and tools such as SLEC [sequential logic equivalence checking], LEC [logic equivalence checking], simulators, and power estimation tools.

**Hasegawa:** We are using HLS with all the designs. Only one high-level synthesis tool fits us. We use SLEC for verification and JEDA for model validation. We are testing the equivalence verification tools between the behavioral source specification and RTL. Currently, we believe behavior-level coverage and assertions are important considerations.

**Chawla:** HLS has evolved from a block-based system methodology to a subsystem design methodology capable of handling multimillion-gate, complex application engines. HLS sits on top of the existing RTL-to-GDS II design methodologies. The high-level synthesis for pure datapath-dominated designs is well defined and works well. Formal verification is a challenge and needs to be better integrated with HLS tools to improve verification productivity.

**Pai:** HLS handled all the complex datapath design challenges present in designing a transactor without the tedium of having to design and debug the complex control path associated with it. It was very easy to import and interface with the legacy RTL blocks, which enabled early architecture modeling and validation. The language doesn’t have the notion of a synthesizable subset. This gives the designer the full freedom to use all the features of the language. This is a major enabler when moving up the abstraction level. The tool flow didn’t change after the specification...
was compiled to Verilog code. This also enabled its quick integration with the existing design flow.

**Koch:** In terms of technology, the change from traditional RTL design to HLS isn’t all that big. There are two main factors involved: the input language has switched to C/C++/SystemC, which are much better suited to express high-level behavior than VHDL or Verilog. Also, behavioral models of designs have always been written in C, but could not be used for synthesis previously. Now they can be used directly, which makes adoption of HLS easier. The second factor is the maturity of the tools. They are much more stable, have a much broader language coverage in terms of supported language constructs, and deliver usable results much more reliably. In addition, complementary technologies have started to become available—for example, formal or co-simulation-based verification technologies to check the functional correctness of the generated RTL against the high-level source description. Such technologies make fitting HLS into the design flow a smoother process.

**D&T:** What was the single biggest design factor that drove you to adopt HLS (verification, power, performance, complexity, time to market, and so on)?

**Trambadia:** Performance. We observed that, for algorithmic designs, the area and timing performance of HLS-generated RTL designs are far better than for handcrafted RTL. HLS has shown a performance improvement of up to 50% in area and timing for a few designs and an improvement of up to 80% in overall product development.

**Clave:** I'd say the biggest factor was time to market, due to the increasing complexity.

**Hasegawa:** Design scalability and verification cost were the major factors.

**Chawla:** Time to market (design productivity) was our main consideration.

**Pai:** The biggest design factor is enabling “early” hardware-software validation and architectural modeling. This was truly made possible by targeting the specification to co-emulation platforms such as EVE’s ZeBu, allowing early hardware-software verification and validation at reasonably high speeds. This approach provides easy porting of untimed transaction-level models in validation. We all know that it is much easier to write untimed TL models in any language. SystemC has an advantage in that it allows you to write “concurrent” untimed models while at the same time unleashing the full power of C++.

**Koch:** The biggest factor for us would be time to market.

**D&T:** What benefits have you seen from using HLS?

**Trambadia:** HLS tools are providing various mechanisms to generate low-power designs, to generate area-optimized designs, and to generate delay-optimized designs. Users can also produce designs that are well balanced between area and delay factors. Important benefits are reduced development cycles and good-quality RTL as compared to handcrafted RTL. What's more, we have also easily achieved the desired frequency (latency) for complex algorithms. HLS has provided design with various performances based on design requirements with the same source code.

**Clave:** A major benefit has been that we could have a smaller development team for a design project of any given complexity. Many scenarios can be benchmarked to select the best architecture. Also, HLS has allowed us to merge software and hardware developers on the same teams, thereby reducing the gap between these two communities.

**Hasegawa:** We have experienced two major benefits: fewer bugs and better algorithm exploration.

**Chawla:** Design productivity is the key benefit for us. Other benefits have been design space exploration for finding the optimal architecture for various target performances and for target ASIC technologies. Another benefit has been high-level design reuse.

**Pai:** The biggest benefit is the speed of integration. We have been able to bring up complex testbenches much faster than before. The hardware acceleration using an emulation platform has allowed us to enable early software development too.

**Koch:** I'd say the major benefit has been an overall increase in productivity. We can build an FPGA
prototype from a C algorithm quickly and then generate RTL for the ASIC target from the same source instantaneously once we are satisfied with the algorithm's performance. Being able to judge algorithm performance in real time is essential in video processing. Previously we either had to do design twice—once for FPGA and once for ASIC—or we used the RTL developed for FPGA for targeting the ASIC technology with a penalty. With HLS, we can use the same source for both without penalty.

**D&T:** With which of the following have you had the most success with HLS?
- Reducing bugs and overall verification burden
- Achieving low-power goals
- Achieving superior design architecture
- Achieving better design reuse
- Shortening your time to market

**Trambadia:** HLS tools that are commercially available are generally used to achieve benefits from all of the above parameters but in our case, I can say we have benefited most with “Reducing bugs and overall verification burden” and “Achieving better design reuse.” HLS tools provide a very wide range of attributes and options, and so by using them we have been able to generate an RTL specification with different area and timing constraints. We resolved most of the functional issues at the behavioral level through behavioral simulation, which reduced the verification burden at the RTL level. Similarly, it is very easy to change design functionality and reuse the design, because we only need to change it at a high abstraction level, which lets us regenerate RTL code with only one click.

**Clave:** Clearly, we’ve achieved the most success in reducing bugs and in overall verification efforts, and then achieving superior design architecture, thanks to the fast architecture evaluation. Moving a large part of the design verification efforts to more abstract languages make DV faster and more exhaustive. High-level synthesis builds “correct by construction” RTL code and reduces the source of bugs, thanks to a more compact functionality coding.

**Hasegawa:** Reducing bugs and overall verification burden is where we’ve had the most success.

**Chawla:** For us, the greatest success has come in shortening our time to market.

**Pai:** Many first-time successes have allowed us to find bugs in hand-coded RTL. It’s been faster than the traditional RTL design flow.

**Koch:** I would say that our greatest success is in achieving better design reuse and shortening the time to market. However, shortening time to market is an indirect effect of using HLS. We’re shortening the time to market through increased productivity and having a reduced verification burden.

**D&T:** Who is the ultimate end user of HLS (system architect, hardware architect, RTL coder, algorithm researcher)?

**Trambadia:** In my view, the RTL coder is the one who is the ultimate end user of HLS as he/she does not need to spend time in modeling complex logic in hardware description language and can easily implement very complex algorithms at the behavioral level, and can convert to RTL code with HLS.

**Clave:** There is no solid job definition for the “ultimate end user of HLS”; I would say this depends on the people skills. There is strong value for algorithm experts, and for system and hardware architects, in using HLS. Overall, HLS breaks down the wall between software developers, and algorithm and hardware designers. Designers with the right skills (C programming experience coupled with a knowledge of high-level hardware architecture, synchronous languages, functional verification) on either side can use HLS and create efficient hardware.

**Hasegawa:** One hardware architect creates the design environment. With many RTL coders, an algorithm researcher uses HLS to create the design and many verification engineers can run simulations. The most important challenge is how to create the design environment.

**Chawla:** I think it would be the hardware architect. Most of the RTL coders would evolve to be hardware architects through using HLS.

**Pai:** System architect and hardware architects would be the primary users of HLS. But we haven’t applied it to other design areas yet.

**Koch:** In our company, the algorithm researcher, system architect, and hardware architect are to some
extent the same person. The algorithm researcher also writes the software model of his algorithm, and he is also the one who verifies the algorithm on FPGAs. With HLS, he's less dependent on the RTL designer because he can do more of the design process himself.

**D&T:** Now that you’ve talked about what has worked, what just plain didn’t work or needs substantial improvement?

**Trambadia:** On the basis of my knowledge and experience with HLS, I feel there are two main issues that need improvement. First, there are a few designs with complex pipelined behavior that are not easy to model at the behavioral level, so to get RTL from HLS for this kind of complexity developers must take care of many things while implementing the design. The overhead is the same as if we implemented the design in RTL. Second, equivalence checking of design behavior between the behavioral and the RTL levels needs improvement. Industry lacks a few strong mechanisms that can ease the equivalence checking of design behavior between those two levels.

**Clave:** We are at the beginning of the learning curve, so from my perspective, deployment and alignment with different usage models across teams and flow integration could be improved. Also, project planning with multiple modules and tools is complex and fairly inaccurate.

**Hasegawa:** One HLS tool is required for all things—something we don’t have today. One requirement is better reduction of redundant logic. Current HLS tools are not powerful enough to create hierarchical designs, but we are working with HLS tool providers to construct these environments.

**Chawla:** In my view, what needs improvement is the ability to define reactive control-based designs along with datapath-oriented designs. This has been the challenge with pure C/C++-based HLS tools. In the verification area, formal (sequential equivalence checking) between C/C++ and generated RTL needs to be improved substantially.

**Pai:** In the case of Bluespec System Verilog (BSV), understanding rule and interface semantics is a complete paradigm shift in how you design hardware. It is like moving from coding in assembly language to coding in C/C++ language. This might be the toughest transition for some folks to make.

**Koch:** Generally, error messaging needs to be a lot better than it is today. In far too many cases, even experienced users cannot interpret scheduling errors properly, nor can they fix them without lots of trial and error or help from the tool vendor.

**D&T:** What about C++ versus SystemC—can they peacefully coexist, or will this be another VHDL versus Verilog war?

**Trambadia:** I can say that C++ and SystemC are both useful, but in different respects. We generally use C++ for ESL [electronic system-level] purposes and for the simulation benefit of nonsynthesizable models. On the other hand, for HLS tools to implement behavioral models that are acceptable to HLS tools, SystemC has now become a primary requirement. So it is not exactly like the VHDL versus Verilog war.

**Clave:** Is a single language to capture everything mandatory? C/C++ has the largest installed base for application engines like codecs and modems, which is challenged now by Matlab and Simulink or symbolic tools.

Also, we think that SystemC is very well suited to, and being heavily used for, verification and system modeling as an integration language whereas C/C++ is more common for algorithmic descriptions. Many HLS tools provide SystemC models as outputs—that is one way to bridge the gap between the two. But C++ is not suitable to describe parallelism explicitly or to manipulate the logical time: for example, when describing protocols or reactive systems. There are other formalisms and languages such as Esterel or BSV, which are superior when describing protocols and parallelism.

**Hasegawa:** I think SystemC is just C++. We are working on the creation of a design environment with C++.

**Chawla:** C++ design description is suitable for datapath-oriented design techniques. What SystemC provides is the notion of a multithreaded simulation and synthesis description that could be used to represent the desired model of explicit concurrency in the system description. A mix of C++ (untimed datapath)
and a SystemC-like language (explicit timed model and explicit concurrency) could evolve to be able to describe both datapath- and control-intensive designs.

Pai: We really think that C++ and SystemC are the best testbench languages, but they come up short as the design languages or design entry languages. This issue goes back to portability between the synthesizable subsets of different vendors’ support for synthesizable subsets and whether they provide the “same” support. BSV, on the other hand, doesn’t face this problem because every construct is synthesizable.

We find tremendous advantage in using SystemC as the concurrent testbench language and BSV as the hardware design language. Verification based on hardware verification languages, such as Vera, SystemVerilog, and so on, appears more as an artifact of using Verilog or VHDL as the design language. This will perhaps give way to C++ or SystemC if instead BSV is chosen as the design language. The size of the verification task is much less, perhaps as much as 75% less. Designers can directly move to validation of the data path and not worry about having to verify the control path because, by design, it is correct.

Koch: I don’t see much of a VHDL/Verilog war, and I don’t see why C++ and SystemC shouldn’t coexist. It’s not a question of C++ or SystemC (both are C++); it’s a question of having a standardized synthesizable subset. The synthesizable language constructs can in both cases be much the same. The difference is that SystemC adds (and sometimes requires) concurrency, timing, and module hierarchy. But once we’re down to writing functional code, there is no difference between C++ and SystemC.

D&T: What do you see for your HLS methodology or flow, looking ahead the next 12 months?

Trambadia: Because of current success in achieving very good performance from IP blocks, we will continue using HLS tools for our next developments.

Clave: We will consolidate the RTL generation with design verification and functional modeling to minimize the number of models and views throughout the development cycle.

Hasegawa: We are satisfied with the methodology of current HLS tools. We are considering how to bind the TLM [transaction level modeling] environment with design using HLS.

Chawla: We need better support for recursive (closed-loop feedback) system design. In addition, we need capabilities to design reactive control along with dataflow designs, and we also need low power as a first-class design constraint for HLS.

Pai: So far, we have only used BSV and that was for synthesizable transactors, but on some of our upcoming projects, we plan to use it for design and moving some more parts of the SystemC testbench into synthesizable transactors.

Koch: That’s a rather difficult question as we are facing a very difficult market situation currently. Therefore, the technical answer I would like to give here is very different from the answer that I am forced to give by the economic situation. Among several other things, we will not be using HLS anymore in the next 12 months.

D&T: Do you think HLS will become widely adopted in the next few years? What could prevent a large adoption of HLS?

Trambadia: Yes, HLS is now becoming more popular among developers and with more EDA companies entering into HLS it will become more generic and easy to use. Obstacles that can prevent the adoption of HLS are developers’ not having a detailed concept of high-level synthesis; fears that HLS tools can replace developers’ jobs, and the high cost of HLS tools.

Clave: Yes, and these factors will be unavoidable with increasing IP complexity. Education is the real bottleneck.

Hasegawa: I agree. We think that knowledgeable hardware engineers must create the design environment with HLS. There are already few knowledgeable hardware engineers who know C++.

Chawla: Yes, but in my opinion, HLS adoption in the future depends on many parameters like standardization of a synthesizable subset; comparable or better quality of results to hand-optimized design; a common ecosystem for modeling, design, and verification;
support for both control- and datapath-oriented designs; and mature formal verification techniques for system-level-model-to-RTL equivalence. HLS also needs to be aligned to the future needs of system-in-package and multicore partitioning and codesign paradigms.

Pai: Every company needs a small group of experts who can drive the methodology so others have an easier time in picking up the proven design flow. This mitigates the adoption risk. Lack of proper training could prevent large-scale adoption. It requires the crossing of disciplines where HLS requires hardware engineers to think like software engineers and vice versa. The “implementation” gap between hardware and software engineers is perhaps the main hurdle for major adoption. It remains to be seen whether large-scale adoption will happen on a live project or in a university classroom. Forming new habits is easier in a classroom than on a live project with various schedule pressures, but at the same time, companies may start taking notice of the productivity gain and find ways for faster adoption.

Koch: I think HLS will become more and more adopted. How widely it will be adopted depends on the ability of the industry to standardize on a common language subset. The more people can switch between different synthesis tools, the more supportive tools will become available, and the more HLS adoption will take place. Also, HLS needs to stop being a point tool and start incorporating classic RTL synthesis into the same tool suite as well, so users can move seamlessly between abstractions and languages, just as they can do today in simulation.

About the participants

Ashish Trambadia is the senior technical lead at NEC HCL System Tech, responsible for IP development and verification methodologies. Gael Clave is a principal engineer responsible for HLS flows definition and deployment, in the methodology group, at Texas Instruments—Wireless Terminals Business Unit.

Hiroyasu Hasegawa is chief technical officer, HD Lab, Japan.

Nitin Chawla is a senior member of the technical staff at STMicroelectronics. He is responsible for the definition of ESL design methodologies for high-performance and low-power applications.

Suhas Pai is a principal engineer working on architectural modeling of 4th-generation chips at Qualcomm Flarion Technologies.

Gernot Koch is the team leader in the Functional Verification CAD group supporting system engineering and design with tools and methodology at Micronas GmbH.

Silver Bullet Security Podcast

In-depth interviews with security gurus. Hosted by Gary McGraw.

www.computer.org/security/podcasts

Sponsored by Security-Privacy

IEEE Design & Test of Computers
Nonmember rate of $32 for S&P magazine!

*IEEE Security & Privacy* is THE premier magazine for security professionals.

Top security professionals in the field share information on which you can rely:

- Silver Bullet podcasts and interviews
- Intellectual Property Protection & Piracy
- Designing for Infrastructure Security
- Privacy Issues
- Legal Issues & Cybercrime
- Digital Rights Management
- The Security Profession

Visit our Web site at www.computer.org/security/

Subscribe now! www.computer.org/services/nonmem/spbnr