**The Evolution Of High-Level Synthesis**

321Shares

 59

 23

 217

*HLS is beginning to solve some problems that were not originally anticipated.*

AUGUST 27TH, 2020 - BY: [**BRIAN BAILEY**](https://semiengineering.com/author/brian/)

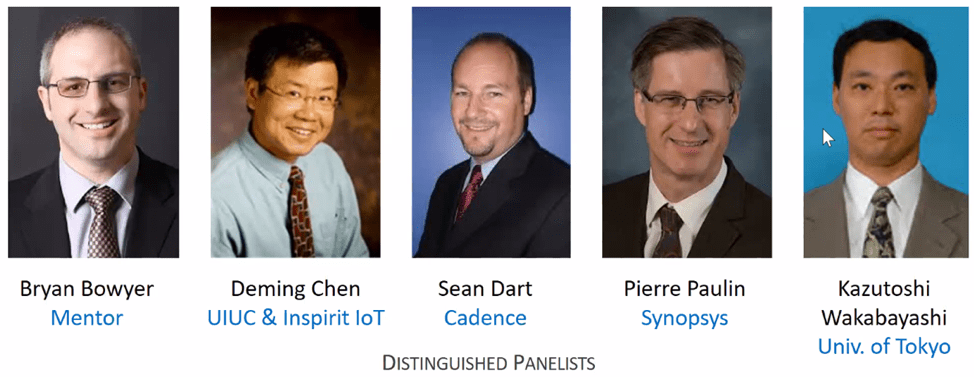
popularity

High-level synthesis is getting yet another chance to shine, this time from new markets and new technology nodes. But it’s still unclear how fully this technology will be used.

Despite gains, it remains unlikely to replace the incumbent RTL design methodology for most of the chip, as originally expected. Seen as the foundational technology for the next generation of [EDA](https://semiengineering.com/knowledge_centers/eda-design/definitions/electronic-design-automation/) companies around the turn of the millennium, [HLS](https://semiengineering.com/knowledge_centers/eda-design/verification/high-level-synthesis/) was partly derailed by the rise of reuse and the IP industry, constrained by the complexity of the algorithms required, hampered by the languages available, and it had an uphill battle to replace the language and methodology that was working adequately for the industry.

None of those factors have changed significantly. But for a certain class of problems, HLS does deliver on the early promise, and its capabilities are growing.

At the Design Automation Conference, a panel of vendors and researchers recounted how HLS had risen from research project to where it is today and provided a glimpse into where it was heading. Semiconductor Engineering went further, asking similar questions to a wider selection of vendors and sampled what the user base thinks of the current state of HLS.



**How we got here**  
The first panelist was Brian Bowyer, director of engineering for [Mentor, a Siemens Business](https://semiengineering.com/entities/mentor-a-siemens-business/). Bowyer provided a glimpse into the problems of the past and some new developments. “There’s a lot of work that’s gone in over the last 40 years to make this happen, starting with behavioral [VHDL](https://semiengineering.com/knowledge_centers/languages/vhdl/) and [Verilog](https://semiengineering.com/knowledge_centers/languages/verilog/), and then later moving into [C++](https://semiengineering.com/knowledge_centers/languages/c-c/) and [SystemC](https://semiengineering.com/knowledge_centers/languages/systemc/),” he said. “Even 20 years ago you could get great results with high-level synthesis, and companies would adopt it and things were great until something didn’t work. Something on the interface wouldn’t work well, or scheduling would move something one cycle later, and suddenly it’s broken and nobody knew why. It was very hard to understand.”

Much has changed recently. “One thing that has really changed in the last four or five years has been the number of open-source, class-based libraries appearing,” Bowyer said. “You go from having to write all of the low-level details to picking up a pre-built class library that has a protocol defined in it, the assumptions and the intent all in one package. That makes it much simpler to use than it has been in any point of the history.”

An example of this is Matchlib from Nvidia. This is a SystemC/C++ library of commonly used hardware functions and components. MatchLib is based on the Connections latency-insensitive channel implementation.

Keeping up with technology scaling was the primary issue for Deming Chen, professor of engineering at University of Illinois at Urbana-Champaign and co-founder of Inspirit IoT. “As long as technology scaling continues, the cost and the complexity of the design grows rapidly. As a result, we are facing some critical challenges. The first is the productivity gap. Human productivity cannot keep up with design complexity, and that gap is widening. The second is the verification predictability gap. For complicated chips, verification takes longer than the design itself. This produces uncertainty in terms of the chip delivery. The third is the quality gap. For [RTL](https://semiengineering.com/knowledge_centers/eda-design/definitions/register-transfer-level/)-based design, people cannot explore different architectural alternatives. As a result, people are really looking for some new design methodology, that can help to really overcome these challenges. That’s how high-level synthesis really came into the picture. A device described in C or C++ benefits from a 5X to 10X code reduction, and 1,000X simulation speed up. HLS works well with a modular design approach. It can produce IPs and with the right interface naturally fit into the IP reuse and creation strategy.”

Sean Dart, senior group director for R&D in [Cadence](https://semiengineering.com/entities/cadence-design-systems/), focused on the quality and flexibility of implementation. “One of the key things is the ability to re-target IP — the same IP — to many different implementation mechanisms. I may want something that’s in a cell phone, I might want something that’s in a base station, and they use exactly the same algorithm. And I want to be able to target that for different performance levels.

“In the early days, we had the assumption that we would be able to win the game with productivity, and that quality of results (QoR) was secondary,” said Dart. “We were wrong. QoR is always under judgment. You’ve got to be able to produce really competitive QoR in terms of area, power, etc. Plus, it’s got to be easy enough for people to use so the adoption can be picked up.”

Programming models was the area that concerned Pierre Paulin, director of R&D for embedded vision at [Synopsys](https://semiengineering.com/entities/synopsys-inc/). “The objective of going beyond RTL was higher productivity through higher abstraction. One of the obstacles was that there was no standardization of a high-level programming language and programming model. The language wars between VHDL and Verilog versus C-based dialect, and then SystemC and SystemVerilog, slowed everything down.”

That limited HLS’ potential. “There was another class of HLS that we call application specific instruction set processors (ASIP),” said Paulin. “The objective is to have the efficiency of hardware, but address flexibility requirements. In the industry we saw two main classes of approaches to address ASIPs. One is taking a standard core, and then adding custom extensions. Companies like Cadence with Tensilica, or Synopsys with ARC, took this approach. Another approach is to do high-level synthesis of an application-specific core. And along with that core, you’re generating the compiler, the assembler, the ISS and the RTL.”

What really excites Paulin is what’s happening in the AI space. “TensorFlow and ONNX are basically the de facto standards, so we only have one programming model, and two flavors of that. From that high-level programming model, automated mapping to complex heterogeneous parallel platform, with multiple cores, multiple processing units within those cores, and complex multi-level memory hierarchy, is becoming really efficient.”

Finally, Kazutoshi Wakabayashi, professor at the University of Tokyo, and who was until recently a senior manager at NEC, identified some historical problems that needed to be overcome. “One problem was the connection with C-based verification. Another issue was education. Even though we had a successful technology, many NEC designers did not want to use it. NEC ordered all new employee take an HLS design class, and this education was so successful that it then saw widespread adoption.

“The code necessary for good QoR and the code needed for simulation is often not compatible. You have to battle with this problem. Many of our customer tried to use existing C code for HLS but that doesn’t work. Coding style is completely different and not compatible.”

**Quality of results**  
QoR was discussed further during the panel. Wakabayashi said HLS tools can produce better area results, especially when they don’t have to meet maximum performance goals.

Bowyer, meanwhile, said he has seen better QoR when power is a concern. “Designers are less experienced with power, and high-level synthesis can produce lower power designs than a designer would.”

The biggest gains come from exploration. “The algorithms have become very sophisticated and the leading tools have good algorithms to do this optimization, but they’re not going to beat a hand coded implementation for a specific microarchitecture,” said Dart. “With HLS you also have the benefit that you can explore. And that’s where you get a massive benefit.”

Paulin agreed. “You can optimally design the wrong component, or you can near optimally design the right component,” he said. “You could potentially optimize the wrong design to death, or you could identify the right design because you have been able to explore.”

This can be especially tricky when processors are involved. “If you define some instructions and you define the behavior and how the data flows through the pipe, then you can do a lot of iterations and design space exploration quite efficiently,” said Zdenek Prikryl, CTO at [Codasip](https://semiengineering.com/entities/codasip-ltd/). “Without it, you have to do a lot of things manually. You have to change the compiler and the RTL. Then you can try to program the RTL using the new compiler.”

That can involve things beyond the processor itself. “When generating the compiler, you are contributing to one of the important things which is the area of memory,” said Roddy Urquhart, senior marketing manager for Codasip. “In most cases where you have a processor subsystem on-chip, the instruction memory will be bigger than the processor itself and contribute more to area and power. By synthesizing efficiently, not just the RTL but also the compiler, you are contributing to managing the overall silicon area and power consumption.”

Bowyer sees another situation arising with some newer designs. “As you move to new geometries, the rules change and suddenly you you may experience power problems or weird routing problems. Also, as people move to these newer ASIC geometries, they start to realize how much worse legacy hardware is compared to building something new.”

**Domain-specific synthesis**  
The discussion about domain-specific high-level synthesis started in the panel and continued off-line. “I don’t believe HLS is the solution for all problems,” said Codasip’s Prikryl. “HLS is domain specific, and tools should focus on some part of problems that it can solve. In our case, we focused on HLS for the processor. We have a description for a processor written in a C-based language and from that we create portions or parts of the processor.”

Security presents some unique challenges. “In part to address inadequacies of synthesized artifacts, we have developed a version of Cryptol, a domain-specific language for specifying cryptographic algorithms, that is capable of synthesizing software, firmware, and hardware IP—implementations and verification harnesses—from formal specifications,” said Joseph Kiniry, principal scientist at Galois. “This gives us a single language and environment in which to reason about properties, to full-blown first-order theorem proving.”

One of the dreams for HLS in the early days was [hardware/software co-design](https://semiengineering.com/knowledge_centers/eda-design/methodologies-and-flows/hardware-software-co-design/). “No one really agreed on a common threading model for hardware and software,” said Bowyer. “Until you have that standardized, and you have that underlying the language, it’s really hard to do hardware/software co-design. But that is sort of happening now in AI. There are tools that will take in a neural network, and they can generate whatever you want. You can get hardware, you can get software, you can map to a GPU.”

The industry is not giving up on hardware/software co-design. “More advanced tooling is required to bridge the gap between the software developer and the hardware implementation,” said Jordon Inkeles, vice president of product for Silexica. “Today, most C/C++ software code is inherently serial and can be performance-limited. Better performance can be achieved by moving software into hardware. Achieving significant software speed-up using hardware requires deep code insights of the algorithm, and data movement to extract and exploit parallelism. HLS compilers enable users to guide the tool with pragmas or directives, but that leaves users on their own to figure out how and where to insert these pragmas. The next phase in HLS adoption will occur when HLS compilers are able to not only provide these insights, but act on them based on user provided constraints.”

The dream of system-level design has not died. “We are pushing for a top-down design flow by using HLS from the system-level,” says Christoph Sohrmann, group manager for virtual system development for [Fraunhofer IIS](https://semiengineering.com/entities/fraunhofer-iis-eas/). “Starting from a SystemC model, which contains the functional description, then refining the specifications to create an architectural description, followed by an HLS step which delivers the bitstream for an Field Programmable Gate Array (FPGA), or a design for integrated circuit floorplaning and ASIC fabrication. The process can be simplified by using parametrizable modules, mapping them into the system-level model architecture in a dynamic way and at the same time creating the test cases for verification. These approaches would increase the traceability and reliability of the design process.”

**Languages**  
The subject that really fires people up is languages. The existing vendors are firmly committed to C, C++, SystemC. “We barely all agreed to do SystemC,” said Bowyer. “You need a lot of tools around high-level synthesis. This isn’t just a tool for synthesizing a design; there are verification tools, linters, formal, there’s all of these other tools that need to agree with synthesis on what this design means. Maybe someday in the future there will be other languages that are more abstract than SystemC or C++ that people use, but in a practical production environment, we’ll probably be here for a while.”

Cadence’s Dart agrees. “One of the key aspects of this is not just about what the synthesis tools can do, and what the user wants to write,” he said. “It’s about the entire environment around designing something that’s going into a chip. And so you can’t forget about things like debug and verification.”

“C has enough abstractions to be productive for processor design, and a lot of engineers are comfortable with it,” said Prikryl. “If you look at the functional languages, it can be hard to learn because it’s a different approach. C is a standard imperative language, and we didn’t want to try something that was not mainstream. C is mainstream in the embedded world, like it or not.”

There have been some other approaches that have not attracted as much attention. “The term HLS seems to have settled into public consciousness to mean code in C/C++ and let the tool produce a micro-architectural design,” said Rishiyur Nikhil, CTO at Bluespec. “Bluespec and Chisel are high-level hardware design languages (HLHDLs), where the designer is focused on explicit description of micro-architecture. The language constructs used to describe micro-architectures include higher-order functions, polymorphic types, extreme parameterization and compositionality, and high-level constructs to describe behavior (guarded atomic actions, object orientation with guarded atomic action semantics, compositionality of guarded atomic actions).”

Academia is calling for a different direction. “There are some new research directions,” said Chen. “These are based on higher-level languages, such as Python or Java.”

“Python is easier to learn compared to Scala and far more productive compared to C/C++,” said Shashank V M, a student at St. Joseph Engineering College in Karnataka, India. “Python is fast replacing C as the first programming language being taught in educational institutes, so the younger upcoming talent is more likely to be comfortable coding and debugging in Python than Verilog/SV/VHDL.”

So does Python have any significant advantages? “Python is a lot easier,” Bowyer said. “It is a safer language to use. It has a lot of guards on it that make it easier for new users to pick up. But from a capability standpoint, I don’t think it gives you any additional capability.”

And there’s more work tied to that. “If you use Python, we have to create a new C++ library for high-level synthesis,” said Wakabayashi.

At the end of the day, it is designers who will make the choice. “Not everyone comes out of college being a C++ guru,” said Dart. “That means you’ve got to be able to get people into the system. The industry is desperate for people who are experts in this space. If you graduate with experience in HLS, and you have expertise in C++, you will get a job in a heartbeat.”

**Conclusion**  
High-level synthesis has become a solid tool in the EDA toolbox. It is seeing increasing usage in areas where new algorithms emerge, or where designers are less constrained by existing IP. There is huge potential in areas such as AI, and increasingly in processor synthesis since the emergence of the RISC-V open ISA.

While academia may be able to create some new tools based on Python, they will be facing headwinds not only with legacy IP and designs, but they will also have to fight the language war and they will have to provide a complete flow with all of the other necessary tools for production quality designs. That may be tough if they do not provide a significant QoR or productivity advantage. The industry always claims that you need a 10X advantage to beat out the incumbent.