High-Level Synthesis: Past, Present, and Future

Grant Martin
Tensilica

Gary Smith
Gary Smith EDA

Editor's note:
This article presents the history and evolution of HLS from research to industry adoption. The authors offer insights on why earlier attempts to gain industry adoption were not successful, why current HLS tools are finally seeing adoption, and what to expect as HLS evolves toward system-level design.
—Andres Takach, Mentor Graphics

We can divide the evolution of high-level synthesis into three generations, plus a prehistoric period: the current generation is the third, and there is the prospect of a fourth generation to come. Chronologically, prehistory was the 1970s; the first generation covered the 1980s and the early part of the 1990s; the second generation, the mid-1990s until early in the 2000s; the third generation, the early 2000s to today (and perhaps the next few years); and the fourth generation will possibly emerge from the current (third) generation.

From a technical and use model viewpoint, the first generation was the era of research and datapath-domain-specific by-products of research. The second generation was the first age of commercial EDA, behavioral-synthesis tools driven by hardware description languages—and it was a commercial failure. The most recent generation is that of C-based high-level synthesis (HLS), which is mostly oriented to datapath applications. The fourth generation may arrive with multidomain HLS tools that provide good results for both control and datapath domains.

In this article, we discuss the history of HLS, primarily from a commercial tool and practical user perspective. For an excellent summary from a research point of view, see the work of Gupta and Brewer[1] and of Coussy and Morawiec.[2]

Prehistory: Generation 0 (1970s)

The 1970s saw some very early research work in both synthesis and high-level synthesis at a time when the only commercial EDA industry products were the physical layout machines offered by companies such as Calma, Applicon, and Computervision. One of the pioneering groups was at Carnegie Mellon University, with early work by Dan Siewiorek, Don Thomas, Mario Barbacci, Alice Parker, and others. This research focused on design specification, simulation, and synthesis at both the RTL and algorithmic level—for example, see the work of Parker and colleagues[3]—using input languages such as ISP (Instruction Set Processor), ISPL (Instruction Set Processor Language), and ISPS (Instruction Set Processor Specification).

The prehistoric work was groundbreaking research but had very little impact on industrial design. Pre-VLSI, and before the emergence of a commercial EDA software industry (as opposed to complete hardware-software EDA systems for mechanical CAD/CAM and for PCB and IC layout), there was no chance to transfer this research to industrial use except possibly in very large electronics companies, most of which were still adopting early CAD systems at the time.

Generation 1 (1980s—early 1990s)

Generation 1 of HLS was primarily a research generation. Many basic concepts of early HLS were explored and a number of seminal research papers written and presented that had a strong impact on the field's development. For example, the work by Pierre Paulin and John Knight at Carleton University produced results in force-directed scheduling for HLS that has, together with other results from the research and conference presentations of their work, been cited more than 1,000 times in subsequent years.[4] Other important work was done by Dan Gajski and colleagues,[5] Giovanni De Micheli,[6] and Raul

0740-7475/09/$26.00 © 2009 IEEE Copublished by the IEEE CS and the IEEE CASS

Camposano and Wayne Wolf,[7] all of whom published books on the topic.

Other domain-specific, DSP-oriented research projects were the Cathedral, Cathedral-II, and succeeding research done by Hugo de Man and others at IMEC in Belgium.[8] This work illustrated two aspects of first-generation HLS research: first, it used a domain-specific input language—in this case, Silage, oriented to describing DSP algorithms—and second, the attempt to commercialize the research ultimately failed, although it took a long time to do so. Indeed, the Cathedral work has set a record for longevity, having been commercialized first in the Mentor European Development Center, next as Frontier Design's A/RT Builder, then in Adelante Technologies, and finally ending up as the OptimoDE tool within ARM. During that long commercialization history, the technology changed input languages, application scope, and user interfaces many times. Time may have run out on this technology; ARM no longer seems to be promoting OptimoDE as a stand-alone technology, although the company might still be using it internally.

Why Generation 1 failed

Although vital in laying the research foundation that found an outlet in later commercial tools, this first HLS generation failed the commercial and use tests. The reasons can be summed up as four: need, input languages, quality of results, and domain specialization.

During the mid-1980s to early 1990s, design technologies for integrated circuits were undergoing significant change. Automatic placement and routing technologies were revolutionizing back-end design. Adoption of RTL synthesis was just beginning. Under these circumstances, when most designers were just learning how to use RTL synthesis effectively or had not yet even started to use it, the idea that behavioral synthesis could fill a design productivity need was an unlikely one.

The second reason Generation 1 failed was that the type of input languages for these early tools presented great difficulties. At a time when most design teams were just moving from schematic capture to standard hardware description languages (HDLs) at the RTL, adopting new and obscure input languages such as Silage for a new and unfamiliar design approach was a considerable hurdle.

The third reason was the inadequate quality of results. These early tools had simple architectures, expensive allocation, and primitive scheduling, and the resulting designs were difficult for designers to accept.

Finally, the domain specialization of some of these early tools—focused on DSP design—was not appropriate for the vast majority of early ASIC designs, which concentrated on control logic rather than dataflow and signal processing. Most processor design was still highly customized, and early ASIC designs focused on integrating rather unstructured random logic.

Generation 2 (mid-1990s–early 2000s)

Generation 2 of HLS was the period in which the major EDA companies—Synopsys, Cadence, and Mentor Graphics—offered commercial HLS tools. As it dominated the RTL synthesis market, Synopsys' excursion into HLS with Behavioral Compiler attracted the greatest interest.[9] However, Cadence's Alta group for system-level design tools offered Visual Architect (based on the SYNT tool from Synthesia), oriented to signal-processing implementations, and Mentor Graphics offered the Monet tool. An overview of this area can be found in the book by John P. Elliott.[10]

Generation 2 attracted a good deal of interest but was a commercial and user failure. The technology was tried out seriously by a number of users but was found wanting in many respects.

Why Generation 2 failed

Several reasons underlie the failure of Generation 2:

- *Mistaken assumptions about who would use HLS.* These tools assumed that the users of behavioral synthesis would be current RTL synthesis users. It turned out, however, that these designers would only switch to new tools that provided substantial improvement in quality of results (area, performance) at the same effort, or substantial reduction in effort with the same quality of results, and with a gentle learning curve. The Generation 2 HLS tools did not satisfy any of these criteria. Attempting to substitute HLS for working (and improving) RTL synthesis failed.
- *Requiring designers to synthesize all the way down to gates.* Instead of complementing RTL synthesis by building a flow that would use both HLS and RTL synthesis, tools such as Behavioral Compiler required synthesizing all the way down to gates. This was not true of some of the other tools, such as Visual Architect.

- *Wrong input languages*. Behavioral HDLs were used as inputs, thus squarely competing with existing RTL synthesis tools and keeping the user base narrowly defined as RTL synthesis designers. Algorithm and software designers did not find the prospect of learning an HDL and a new set of tools very inviting, and if they tossed their algorithm specs over the wall to RTL designers, then HLS would not be used in any case.
- *Poor quality of results*. In general, HLS results were often widely variable and unpredictable, with poorly understood control factors. Learning to get good results out of the tools seemed a poor investment of designers' time.
- *Hard-to-validate results*. RTL code resulting from HLS might produce the same outputs as the behavioral model but in significantly varying time intervals (that is, which cycle outputs would be produced was highly variable). This made it hard to validate that correct synthesis had occurred. And, of course, no formal methods were available at this time.
- *Not enough attention paid to intermediates and interfaces*. Block synthesis often made poor use of memory and registers to store intermediate results and could not optimize across block boundaries.
- *Flow issues*. Cadence Alta Visual Architect was driven by dataflow libraries as a link to implementation, but use of atomic actor dataflow models often led to poor quality of results. (For example, for 2D visual data, one block would write out a 2D array row-wise, but the next block would read it column-wise, resulting in huge intermediate tokens and a poor use of memory.)
- *Failure to recognize the difference between dataflow and control, and serious marketing overhype (especially of Behavioral Compiler)*. RTL synthesis could be used for both dataflow and control, but HLS produced poor results for control-dominated branching logic as opposed to data-crunching algorithms. (Prior to the early 2000s, designers often used specialized datapath compilers in conjunction with RTL synthesis, especially to better optimize the physical layout). When designers applied HLS to control, the results they obtained were poor, and this discredited the tools. Dataflow and signal processing produced better results, but this was a niche market and the overmarketing of some of these tools overshadowed any good results.

In Generation 2, simulation times were almost as long as with RTL synthesis, since HDL was used for input specification, and HDL simulation was used in the design process. Moreover, HDL use meant that it was impossible to exploit advanced research in compiler-based language optimization (for example, various advances in C compilation, especially across module boundaries). Finally, Generation 2 tools failed to expand the market for HLS by not offering tools to algorithm and software developers.

Generation 3 (from early 2000s)

The third-generation tools include those that have been offered by several vendors since the early 2000s. The list of vendors is rather long, and some of the tools have already moved from one company to other. Among these tools are Mentor Catapult C Synthesis, Forte Cynthesizer, Celoxica Agility compiler (sold in 2008 to Catalytic, which renamed itself Agility Design Solutions), Bluespec, Synfora PICO Express and Extreme, ChipVision PowerOpt, NEC CyberWorkBench, AutoESL AutoPilot, Xilinx AccelDSP (which started as the product of an independent company, AccelChip) and SystemGenerator, Esterel EDA Technologies Esterel Studio, Synopsys Synplicity Synplify DSP, and, just announced in the summer of 2008, Cadence C-to-Silicon (C2S) compiler. Recent changes in the market include Mentor's acquisition of the Handel-C synthesis technology from Agility Design in January 2009 (this was the ex-Celoxica Agility compiler). Some of these tools primarily target ASIC and ASSP (application-specific standard product) design; others primarily target FPGA design, and still others target both. Some are very clearly aimed at the dataflow and DSP domain, a few at control (for example, Esterel Studio), while others claim capabilities suitable for both dataflow and control (for instance, Bluespec and Cadence C-to-Silicon). Most of these tools use a variation of C, C++, or SystemC as input (for example, Celoxica used a special dialect of C called Handel-C), but this is not universal. Esterel Technologies used graphical state machine capture and Esterel as an intermediate language. Bluespec has relied on certain constructs in SystemVerilog (after using the Haskell programming language as input when it was an MIT research project), although they also offered a SystemC input mechanism. Some of the DSP-oriented tools are driven by the MathWorks Matlab and Simulink software, although potentially using C as an intermediate format.

The ability to handle both dataflow and control domains is arguably a characteristic of future, fourth-generation HLS tools, and thus some of the third-generation tools may be evolving to offer fourth-generation capabilities.

Why Generation 3 is succeeding

Although HLS lacks publicized comparative benchmarks for the various tools, and there are few industrial successes that have been objectively written about (free of vendor marketing), indications are that Generation 3 of HLS tools is achieving some reasonable success, especially in Japan and Europe. (Of course, tool vendors often restrict publication of benchmarking results, and legacy issues can make comparisons more difficult.)

There are several reasons why this generation of tools is doing better than the second generation.

- *Focus on domain of application.* With some exceptions, most of these tools are focused on dataflow and DSP design domains, and achieving reasonably good quality results there. By not being overly hyped as suitable for all design styles, the tools are being applied in domains where they are expected to have a higher probability of success.
- *Focus on algorithm and system designers with the right input languages*. With only a few exceptions, by focusing on dataflow, DSP, and algorithm designers, and offering them the languages with which they are comfortable (C variants and Matlab), the tool vendors have correctly identified a good market niche that wants to quickly explore and implement complex algorithms into either ASIC or FPGA forms. The vendors are not asking these designers to learn unfamiliar HDLs and unusual language constructs beyond simple pragmas.
- *Improved quality of results.* This generation of tools can take advantage of research into compiler-based optimizations, driven as it is by variations of software languages rather than relying only on HDL-driven improvements. By all accounts, these tools are enabling designers to achieve improved design outputs.
- *Design domains that have shifted.* Many products are incorporating significant amounts of signal and multimedia processing, and require blocks to be implemented in hardware to accelerate portions of these algorithms. Thus, a greater proportion of designs can take advantage of at least some HLS.
- *The rise of FPGAs.* When a team wishes to quickly map an algorithm into an FPGA implementation, the measurement criteria are different than they are for an ASIC or ASSP. The design has to "fit" into the FPGA—since FPGAs come in discrete sizes, the size in which the design fits might have many spare gates. And the design has to work fast enough—but if this is well within the FPGA speed and size capacity, the size-speed trade-off is different than for ASICs. Using high-level synthesis with FPGA targets is a perfect way of quickly getting an algorithm into hardware.

The chart in the sidebar "High-Level Synthesis: Commercial Progress" illustrates both the rise and fall of Generation 2 HLS tools, and the current rise of Generation 3.

Who is using Generation 3?

Obtaining detailed information on the usage of EDA and electronic system-level (ESL) tools is not easy: it often takes many years before enough meaningful user experiences are documented in technical papers, and it also has been tool vendors' practice to restrict their users from publicly discussing comparative benchmarks. So to get an idea of where Generation 3 HLS tools are being used, we must rely more than we would like on tool marketing information. Nevertheless, we can derive some useful information from these sources. In addition, some user comparative experience is reported on websites such as John Cooley's DeepChip (http://www.deepchip.com).

Examining technical publications from Mentor Graphics on Catapult C Synthesis, we find Nokia using HLS to generate hardware implementations of DSP algorithms for wireless communications; the flow started with floating-point algorithms captured in Matlab, and included blocks for clock tracking, frequency offset correction, and equalization. Alcatel Space also applied Catapult to DSP blocks for power, frequency, and timing recovery. Ericsson evaluated Catapult on DSP blocks on three different design projects: wideband code-division multiple access (W-CDMA) third-generation modem design; on implementations of inverse discrete cosine transforms (IDCTs) in 2D graphics applications; and on algorithms for GSM (Global System for Mobile Communications), the mobile phone standard. Fujitsu used Catapult on communications algorithm blocks. Finally, Toshiba used Catapult for QR decomposition using alternatives to Cordic (for

High-Level Synthesis: Commercial Progress

In tracking the commercial adoption of high-level synthesis, we find that HLS adoption mirrors the technology's various phases. The years 1994 to 1996 were the experimental years (see Figure A); 1997 to 1999 were the years of Synopsys' Behavioral Compiler, and of its rivals from Cadence and Mentor Graphics. This is Generation 2 as described in the main text.

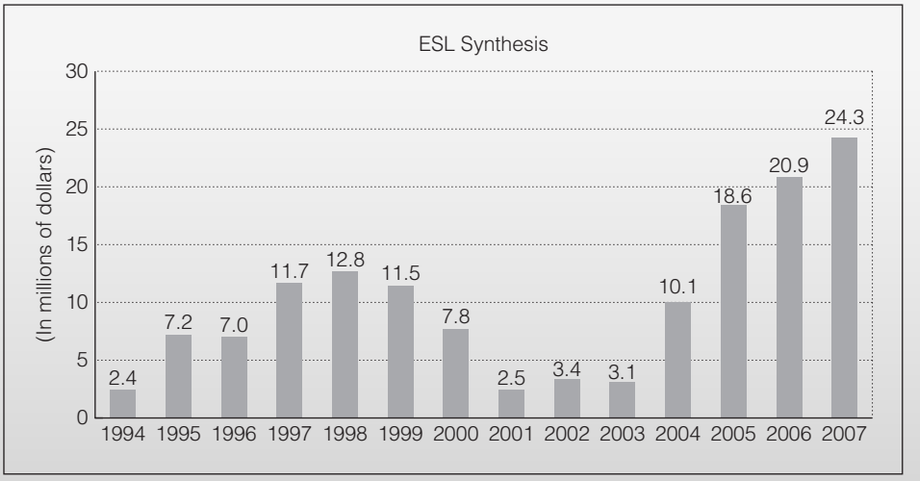

Figure A. Sales of electronic system-level synthesis tools. (Source: Gary Smith EDA statistics.)

The Behavioral Compiler was actually the first of the fast algorithmic design compilers, and had a good reputation with that market segment. Synopsys' problem was that it oversold the product; consequently, the negative reports from ASIC designers, who should never have purchased the product in the first place, killed it. That negative view of HLS coupled with the 1990s' recession resulted in the low sales volume numbers in 2001 to 2003.

The fast algorithmic design compilers, now called C to RTL or high-level FPGA compilers, were responsible for the market taking off in 2004, as Generation 3 emerged. The years 2004 to 2007 have been fairly evenly split between the algorithmic C compilers and the SoC SystemC compilers. It will be interesting to see the interplay between these completely different markets, and whether a fourth generation will emerge in the next few years.

As of 2008, the market for HLS tools sees Mentor Graphics as the leader in the C to RTL SoC design market, and in the C to FPGA market. Forte leads in the SystemC to RTL area for SoC design. As noted in the main text, Mentor Graphics acquired the Agility Design Handel-C to FPGA synthesis technology in January 2009, which strengthened it in the C to FPGA area. With its showing in both the SoC and FPGA HLS domains, Mentor Graphics is the overall HLS market leader.

*co*ordinate *r*otation *di*gital *c*omputer) algorithms for multiple-input, multiple-output communications—again, more DSP-style design.

From Forte's website, we see acknowledgments that Toshiba used the company's Forte Cynthesizer for H.264 multimedia design, Seiko-Epson used it for communications devices, Oki for system LSI, and Sanyo for consumer electronics.

When Cadence announced its C-to-Silicon compiler in summer 2008, it cited Hitachi and Renasas as early users, albeit without identifying specific product domains.

John Cooley's Deepchip website had some comment on the relative advantages of Mentor Catapult C Synthesis and Forte Cynthesizer: in a posting dated 23 June 2006, users from Prime Gate Ltd., STMicroelectronics, Nokia, Casio Computer, Motorola Labs, Pioneer, and some anonymous companies commented on Catapult, Cynthesizer, or both tools. Applications reported included image processing and dataflow, wireless receiver algorithms, JPEG encoding, and YUV transformation. In general, this (very limited) set of comments revealed a lot of support for Mentor Catapult C.

More recent postings on Deepchip (for example, 20 November 2008, and 2 February and 2 April 2009) include some active commentary on a variety of HLS tools, including vendor discussions of Mentor Catapult C Synthesis and Synfora PICO, and user experiences with Catapult C (two anonymous discussions) and Cadence C-to-Silicon versus other HLS tools (Micronas). All the user experiences discussed

were on realistic design benchmarks or real designs, and the write-ups included both good features of the latest wave of tools as well as areas for improvement. The net conclusion of all three users was that, going forward, HLS would be a part of their design flow. In general, the past couple years have seen many positive user experiences with Generation 3 tools, and real adoption of them.

Summarizing these user experiences, we can see a lot of interest in using HLS for DSP blocks for wireless and wired communications and for image processing. In Japan, there's been particularly strong interest in the technology. The emphasis on DSP fits in well with the overall perception that third-generation HLS is particularly suitable for signal-processing domains.

As we've mentioned, some third-generation HLS tools claim good effectiveness in both dataflow (algorithm) and control. We can also see synthesis tools coming from the configurable-processor domain, such as the extensive research by Paolo Ienne, Laura Pozzi, and colleagues[11] and the Tensilica XPRES tool[12] that map C into gates using RTL synthesis. The form of the gates, however, is in terms of a configured extended processor. We also see technologies being applied to both ASIC and FPGA styles.

All of this points to the rise of a possible fourth-generation multidomain HLS toolset. However, none of these tools is comprehensive in targeting all the domains: control and dataflow; FPGA and ASIC; random logic and processor-based; and hardware, software, and mixed hardware-software forms. As a result, these tools do not let designers explore the design space in even a semiautomated format without changing toolsets and investing a lot of work in tracking and optimizing various branches of the design tree. If someone is capable of offering a true design space exploration and implementation tool that covers all targets and forms with effective synthesis algorithms, we would then be on the threshold of true system-level synthesis: a longtime dream, but one that is not yet close to full realization. That would make the fourth generation of HLS tools a generation truly to be reckoned with—and one that would lead to radical changes in the future mainstream design technology. ■

References

1. R. Gupta and F. Brewer, "High-Level Synthesis: A Retrospective," *High-Level Synthesis: From Algorithm to Digital Circuit,* P. Coussy and A. Morawiec, eds., Springer, 2008, chapter 2.
2. P. Coussy and A. Morawiec, eds., *High-Level Synthesis: From Algorithm to Digital Circuit,* Springer, 2008.
3. A. Parker et al., "The CMU Design Automation System—An Example of Automated Data Path Design," *Proc. Design Automation Conf.* (DAC 79), ACM Press, pp. 73-80.
4. P.G. Paulin and J.P. Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASICs," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 8, no. 6, 1989, pp. 661-679. According to Google Scholar, consulted 13 September 2008, this paper has been cited 648 times. Other HLS-related papers by these authors, in *Proc. DAC* 1986, *Proc. DAC* 1987, *Proc. DAC* 1989, and *IEEE Design and Test* 1989 have been cited another 462 times according to Google Scholar. A more conservative test using Microsoft China's Libra database finds 339 citations for the same set of papers.
5. D. Gajski et al., *High Level Synthesis: An Introduction to Chip and System Design,* Kluwer (now Springer), 1992.
6. G. De Micheli, *Synthesis and Optimization of Digital Circuits,* McGraw-Hill, 1994.
7. R. Camposano and W. Wolf, eds., *High-Level VLSI Synthesis,* Springer, 1991.
8. H. de Man et al., "Cathedral-II: A Silicon Compiler for Digital Signal Processing," *IEEE Design and Test,* vol. 3, no. 6, 1986, pp. 13-25.
9. D. Knapp, *Behavioral Synthesis: Digital System Design Using the Synopsys Behavioral Compiler,* Prentice Hall, 1996.
10. J.P. Elliott, *Understanding Behavioral Synthesis: A Practical Guide to High-Level Design,* Kluwer (now Springer), 1999.
11. L. Pozzi, M. Vuletic, and P. Ienne, "Automatic Topology-Based Identification of Instruction-Set Extensions for Embedded Processors," *Proc. Design, Automation, and Test in Europe* (DATE 02), IEEE CS Press, 2002, p. 1138.
12. D. Goodwin and D. Petkov, "Automatic Generation of Application Specific Processors," *Proc. Int'l Conf. Compilers, Architecture, and Synthesis for Embedded Systems* (CASES 03), ACM Press, 2003, pp. 137-147.

Grant Martin is a chief scientist at Tensilica, Santa Clara, California. His interests include electronic system-level design, platform-based design, embedded software, and embedded systems. He has an M.Math from the University of Waterloo, Canada. Grant is a senior member of the IEEE.

Gary Smith is the founder and chief analyst of Gary Smith EDA. His interests include EDA, ESL, and electronic design methodology. He has a BS in engineering from the United States Naval Academy. He is a current member of the Design TWG for the International Semiconductor Road Map (ITRS) and editorial chair of the IEEE Design Automation Technical Committee (DATC), and he serves on the DAC Strategic Committee.

■ Direct questions and comments about this article to Grant Martin, Tensilica Inc., 3255-6 Scott Blvd., Santa Clara, CA 95054-3013; gmartin@tensilica.com.