Productivity Issues in High-Level Design: Are Tools Solving the Real Problems?

Reinaldo A. Bergamaschi
IBM Research Division
Thomas J. Watson Research Center
Yorktown Heights, N.Y., U.S.A.

1 Introduction

For the last decade, high-level design has been considered the breakthrough technology that will enable designs to be completed in a fraction of the time of current methodologies. This notion has been put forward by researchers in academia and industry. While this concept is perceived to be true, it is unclear how much time is actually saved in the total design cycle, including specification, synthesis, verification (simulation), layout and test.

This paper analyzes the impact that a high-level design methodology can have in the total design cycle. The main question being raised here is how much high-level design tools really help. For example, one can use a high-level tool to perform scheduling or generate a pipelined implementation of a *for loop*; and that may save a week in design time. But that alone does not decrease the months of simulation time that still must be performed in large designs. This paper discusses these issues and presents possible approaches for making high-level design tools more effective in the total design cycle.

The term *high-level design tools*, as used in this paper, denotes primarily synthesis and analysis tools capable of handling hardware description languages at the behavioral and register-transfer (RT) levels.

2 High-Level Design Methodology Overview

The methodology analyzed in this paper is shown in Figure 1. Typically, a design starts with some high-level (e.g., English) specification that gradually gets refined to a hardware description in a mixture of abstraction levels. A complete design normally incorporates pieces described at the behavioral, RT and gate level as well as custom partitions (transistor level).

32nd ACM/IEEE Design Automation Conference ®

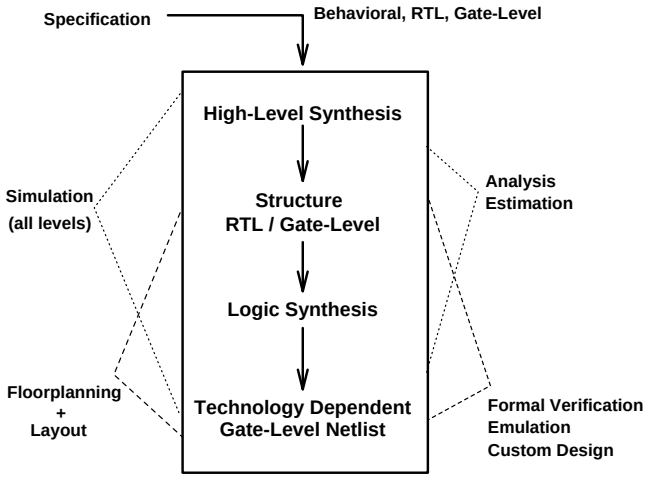

Figure 1: High-level design methodology.

This refinement process is usually done manually, although there are modeling tools available in the market that can to help automate it.

The complete synthesis path can be divided into two parts: first, high-level and RTL synthesis map a language description into a structural representation at the technology independent RT level or gate level; second, logic synthesis performs optimization and technology mapping producing a technology dependent gate-level netlist. The intermediate RTL structural representation is ignored in most high-level methodologies as tools operate mainly at the two extremes, language and gate-level descriptions.

Simulation starts as soon as a hardware description is available. The type of simulator used (e.g., event-driven, cycle-based, gate-level) is dependent on the simulation tools and on the level of description being simulated. Thus, if behavioral VHDL is available, an event-driven simulator is used. If an RTL network is available, a cycle-based simulator can be used. As these simulators differ in performance, it is beneficial to use the fastest simulator as early as possible in the design process. Currently, cycle-based simulators can outperform event-driven simulators by an order of magnitude or more.

Other design steps, such as area and delay estimation, floorplanning, emulation and formal verification operate usually at the gate-level.

The underlying problem with this methodology is that tools are not working on the most efficient level of design. For example, cycle-based simulators are preferable to event-driven ones for performance reasons. But they need RTL structural representations which are not usually available (language level and gate-level are the common formats). Estimation and floorplanning tools either operate at the top, block level representation, with corresponding inaccuracies, or at the gate level which is accurate but may have a much larger size and impact performance. Emulation and formal verification tools also use gate-level and netlists.

High-level synthesis represents the bridge between hardware descriptions at the behavioral and RT-level and the lower-level tools in existing methodologies which require structural networks as input. Most high-level synthesis systems function as compilers that map language descriptions into RTL networks, performing various optimizations, such as scheduling, allocation, binding, etc. That is, high-level synthesis typically operates as a front-end optimizer for logic synthesis; in this role, its value-added is limited because it does not help significantly to speed up the design process. Behavioral descriptions can be significantly more compact than gate-level or even RTL descriptions. However, specification time is only a fraction of total design time, which in high-performance designs is dominated by simulation and logic synthesis.

High-level synthesis can be a lot more effective if used as a front-end optimizer for all other design tools that require structural networks as input. Significant design time can be saved if the designer can, in the much shorter time that it takes to run high-level synthesis, obtain a structural RTL/gate-level representation which can be used for estimation, simulation and formal verification. This requires changes in most tools. First of all, high-level synthesis needs to be fast, efficient and tuned to the other tools. For example, it is important that the network produced by high-level synthesis be efficient for simulation purposes. Hence, it is better to produce an RTL network with multi-bit operators and busses, than to use a gate-level network. However, estimation of area and delay works better at the gate-level, thus new estimation algorithms at the RT-level need to be developed to take advantage of the RTL network produced by high-level synthesis.

3 Can High-Level Synthesis Deliver?

The approach presented in the previous section can only work if high-level synthesis systems become more efficient and more acceptable to designers. Designers are clearly interested in tools that help them *solve* problems, and as they see it, high-level synthesis is, in fact, creating or ignoring the following problems:

- **Register problem:** high-level synthesis may create registers due to scheduling or inferencing. The creation of *unexpected* registers affects the simulation and verification steps. Designers need to know how and why registers are being created and need control mechanisms to guide synthesis.

- **Simulation problem:** if scheduling is performed, the cycle-by-cycle behavior of the synthesized design may differ from the behavior specified in the initial description. This may cause the simulation results after synthesis to mismatch (on a cycle basis) the language simulation results. This may be not be a problem in designs whose function is to compute a result at the end of several cycles, however, it is a significant problem in designs that perform a lot of handshaking and output results at different cycles.

- **Estimation problem:** most of the estimation work in high-level synthesis concentrates on lower/upper bounds on the numbers of basic blocks such as registers, muxes, adders, etc. These metrics are too inaccurate for any practical use. Area estimation has to be based on full control and data-path area. Delay estimation has to be performed at the bit level (worst case block-level timing is too inaccurate). Efficient algorithms are needed that can estimate area and delay at the bit-level without having to expand the network into single-bit operators (which would increase its complexity). This estimation is relatively strighforward for data-path operators, however, it can be very inaccurate for control logic.

- **Efficiency problem:** there are efficient algorithms for isolated problems, such as, scheduling, allocation and resource sharing; however, solving each of these problems well does not necessarily correlate with smaller chip area or delay. This is usually caused by the fact that these algorithms use cost-functions which are not fully representative of real area and delay. The execution time for high-level synthesis is considerably shorter than that of logic synthesis, but it is still not fast enough for it to be used as a front-end for simulation, for example, where model-build time is critical. Ideally, high-level synthesis execution times should approach those of optimizing language compilers which are currently around 5 to 10 thousand lines of code per minute.

- Controllability and Predictability problem: controlling and predicting the output of high-level synthesis is an area that has been largely ignored. Designers, however, need to understand the results of synthesis and need to know how to change and annotate the input description in order to get the desired results. High-level synthesis systems need to provide control mechanisms and generate results which are *predictable* in the designer's view.

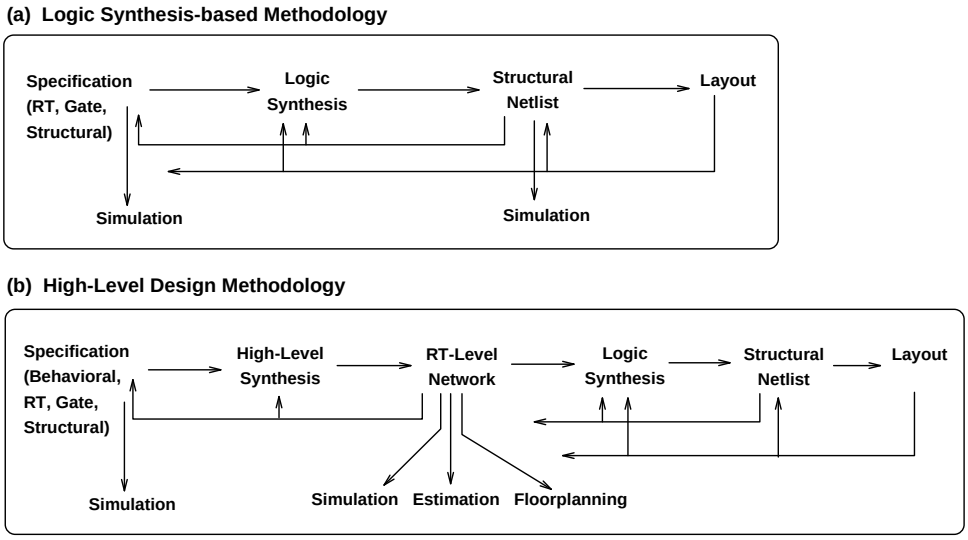

Figure 2: Two design methodologies: logic synthesis-based *vs.* high-level design

- Language subset problem: high-level synthesis systems tend to accept only behavior-level descriptions or even graph representations. A complete design, however, incorporates partitions specified at different abstraction levels, thus high-level synthesis systems need to accept all abstraction levels and be able to apply different synthesis algorithms to each level. Moreover, support for complex language constructs, such as, records, arrays, generics and generates (in VHDL), allows for very compact descriptions which decrease specification time.

Once these problems are overcome, high-level synthesis can start playing a major role as a front-end optimizer for various other design tools and not only for logic synthesis.

4 Savings in Design Time

High-level synthesis can become much more important by providing the designer with relevant design information earlier in the design cycle. Such information can then be used for simulation, verification, redesign, etc., allowing problems to be detected early, saving costly design iterations. To estimate how much time this methodology can save in a high-performance design, one needs to understand that a design is hardly a linear sequence of steps and usually there are several iterations through simulation, synthesis and layout. It is clearly beneficial to minimize the time through each iteration.

Table 1 compares the execution times of high-level and logic synthesis in two small examples using IBM *HIS* System [1] for high-level synthesis and IBM *BooleDozer* [2] for logic synthesis. It is reasonable to expect high-level synthesis to run 10 to 100 times faster than logic synthesis.

	VHDL (lines)	High-Level Synthesis	Logic Synthesis	Size (gates)
Design1	400	17 sec	500 sec	1800
Design2	1500	26 sec	4300 sec	11100

Table 1: Execution time comparison between high-level synthesis and logic synthesis

To estimate the productivity impact from using high-level synthesis, consider the two methodologies shown in Figure 2. Figure 2a shows the prevailing logic synthesis-based methodology (LS), whereas Figure 2b shows a high-level design methodology (HLD) which uses high-level synthesis as a front-end to logic synthesis and other design tools.

The savings in design time in a high-level design methodology depend upon the type of design and on the number of design iterations required, which in turn depends on the performance and size of the design.

An average design cycle of a low-performance design could exhibit the following characteristics:

- Minor functional problems that require some recoding.
- Minor area and timing problems.
- Few synthesis iterations/reruns.
- 4 to 8 logic synthesis runs (in LS methodology) *vs.* 5 high-level synthesis + 3 logic synthesis runs (in HLD methodology).

Under these assumptions, the savings in synthesis time can be expected to be as shown in Figure 3a. A low-performance design is more amenable to be described

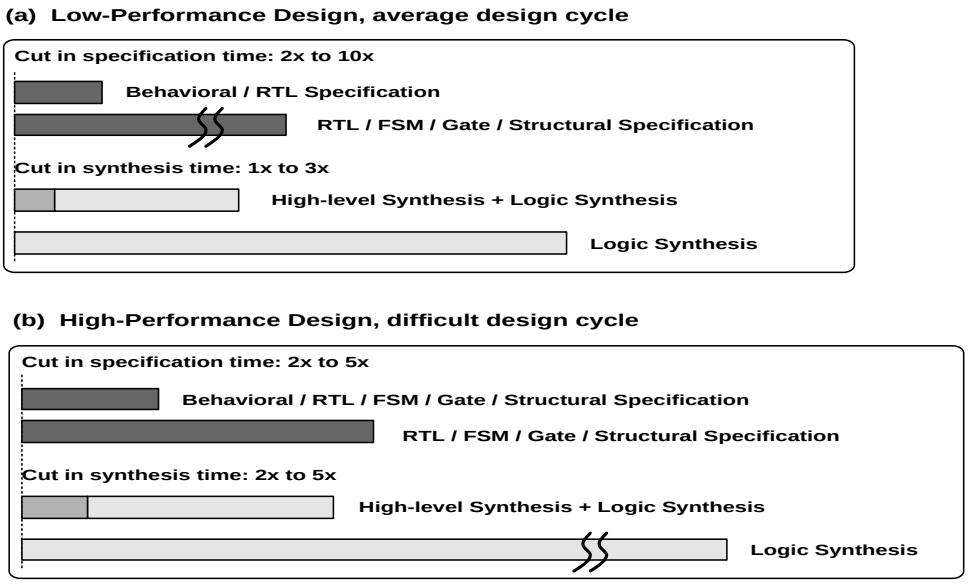

Figure 3: Savings in design time.

using behavioral or RT levels, as less implementation details need to be given. This can save 2 to 10 times in specification time.

Fewer logic synthesis runs are needed in an HLD methodology because many area and timing related problems can be detected through early estimation after high-level synthesis. Cycle-based simulation can also start earlier with the network produced by high-level synthesis. In low-performance designs, logic synthesis will not need to be run several times, therefore the savings in synthesis time in an HLD methodology will be small.

A difficult design cycle of a high-performance design could have the following characteristics:

- Several functional problems.
- Specification changes in the middle of the design.
- Tight area and timing requirements.
- Several synthesis iterations/reruns needed.
- 10 to 30 logic synthesis runs (in LS methodology) *vs.* 10 high-level synthesis + 5 logic synthesis runs (in HLD methodology).

Under these assumptions, the savings in synthesis time can be expected to be as shown in Figure 3b. A high-performance design is likely to have partitions described at lower levels for performance reasons. This reduces the savings in specification time. However, it is even more important that area and timing problems be detected early through estimation. Links to early floorplanning can estimate global capacitances which can then be passed to logic synthesis. This helps decrease the number of post-layout (after backannotation) synthesis runs, thus contributing to bigger savings in synthesis time.

5 Conclusions

High-level design is an emerging methodology based on high-level synthesis which has great potential for speeding up the design process, For this productivity increase to become a reality, high-level synthesis needs to be made more efficient as a synthesis tool as well as more integrated with other design tools. This paper outlined the main problems with current high-level synthesis systems and indicated approaches for using high-level synthesis as a front-end optimizer for other design tools that require structural networks as input, such as estimation, early floorplanning, cycle-based simulation and verification.

References

[1] R. Bergamaschi, R. O'Connor, L. Stok, M. Moricz, S. Prakash, A. Kuehlmann, and S. Rao, "High-level synthesis in an industrial environment," *IBM Journal of Research and Development*, vol. 38, January 1995.

[2] D. Brand, R. Damiano, L. Van Ginneken, and A. Drumm, "In the driver's seat of BooleDozer," in *Proceedings of the IEEE International Conference on Computer Design*, pp. 518–521, IEEE, October 1994.