Challenges and Opportunities of ESL Design Automation

Zhiru Zhang
AutoESL Design Technologies, Inc.
Los Angeles, CA
zhiruz@autoesl.com

Deming Chen
Department of Electrical and Computer Engineering
University of Illinois, Urbana-Champaign
dchen@illinois.edu

ABSTRACT

System-level synthesis compiles a complex application in a system-level description (such as SystemC) into a set of tasks to be executed on various processors, or a set of functions to be implemented in customized logic, as well as the communication protocols and the interface logic connecting different modules. Such capabilities are part of the so-called electronic system-level (ESL) design automation. ESL design automation has caught much attention from the industry recently. In general, it has been shown that the code density and simulation time can be improved by 10X and 100X, respectively, when moved to system level from RTL. Such an improvement in efficiency is much needed for design in the deep submicron era. This paper identifies a set of key challenges in ESL design automation with major focus on high-level synthesis (HLS). We shall discuss existing and potential solutions to these challenges and outline research opportunities and directions in the evolution of ESL design automation.

1. Introduction

The rapid increase of complexity in System-on-a-Chip (SoC) design urges the design community to raise the level of abstraction beyond RTL. Electronic system-level (ESL) design automation has been widely identified as the next productivity boost for the semiconductor industry. However, the transition to ESL design will not be as well accepted as the transition to RTL in the early 1990s without robust analysis and synthesis technologies that help designers quickly converge to high-quality architectures and automatically generate highly optimized implementations.

High-level synthesis (HLS) [8], in particular, is a key cornerstone of ESL design automation. It enables automatic generation of optimized hardware from high-level programming languages and allows effective exploration of software and hardware architectures. Given its huge potential value, numerous efforts on research and development of HLS tools have been conducted in both academia and industry in the past three decades [17]. In the 1980s and the 1990s, the success of older-generation tools was fairly limited in practice. Poor input language selection (e.g., HDL-based) and inferior quality of results (QoR) to manual designs are among most notable reasons for the failure. Since early 2000s, the newer generation of C-based synthesis solutions is gaining much more attractions in the market. The offerings of high-level programming languages to manage the daunting design complexity and the improved QoR are among the major factors that incentivize the industry adoption. Nevertheless, HLS is still an evolving technology. Many synthesis and optimization problems need to be addressed in its way to becoming an imperative step in the mainstream design flow.

In this paper we identify a number of key challenges and research opportunities for ESL design automation (including modeling, analysis and synthesis) with a major focus on HLS. The remainder of our paper is organized as follows: Section 2 discusses the modeling dilemma with the disconnection between high-level simulation model and the synthesizable model. Section 3 presents challenges and promising directions in high-level memory synthesis, power optimization and variation-aware synthesis. Section 4 provides the concluding remarks.

2. Modeling Challenges

Some of the most widespread uses of ESL models and tools in industry today are for early embedded software development, architecture modeling, design space exploration, and rapid prototyping. In particular, transaction-level modeling (TLM) with SystemC [11] has become a very popular approach to describing virtual software/hardware platforms which model large-scale SoCs with multiple microprocessor cores, software stacks, hardware accelerators, hierarchical bus networks, and many other digital and analog IP blocks.

HLS fits in nicely in the context of architecture exploration and rapid prototyping. In architecture exploration, HLS tools can provide quick estimations and analyses of performance, area and power of the synthesized modules. This allows system architects explore different architectures and select best one among them without going through the time-consuming manual process to implement RTLs. In rapid prototyping, HLS tools also provide automated flows to map the ESL models to an FPGA-based system for system emulation, functional validation and real-time debugging.

Note that the TLM models for virtual platform modeling are typically written with a great deal of emphasis on code reusability and readability and are heavily optimized for simulation speed. Hence C++/SystemC-based TLM models often make use of complex pointer schemes (e.g., pointer maps) and sophisticated object-oriented features such as run-time polymorphism (e.g., virtual member functions) and dynamic pointer/memory management (e.g., smart pointers).

However, not all of these features are efficient or even feasible for hardware synthesis[1]. Due to the static nature of hardware, the usage of pointers is typically restricted (to compile-time determinable ones) and dynamic memory allocation/deallocation is generally forbidden. Therefore, the conversion of simulation-

[1] The HLS synthesizable subset remains to be vendor-specific for the time being.

oriented virtual platform specification into a synthesizable specification remains to be a manual process as illustrated in Figure 1.

Designers need to ensure that the synthesizable code is functionally equivalent to the original version. They also need to maintain two sets of ESL models and keep them in sync.

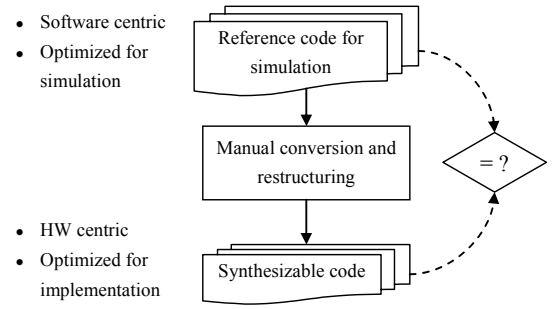

Figure 1: Manual code modification for hardware synthesis.

It would be ideal for the designers to maintain a single synthesizable model as the golden reference for both simulation and synthesis. To achieve this goal in the near future, the ESL community shall make progress in the following areas:

- The synthesis tool shall continue to improve to handle a broader class of language constructs. It is particularly desirable to support dynamic behaviors in certain restricted forms. For instance, the Pinapa front end with hybrid parsing and elaboration capability [16] demonstrates one promising method to extract the static SystemC binding and connectivity from the seemingly dynamic specifications. Similar approaches can be taken to extend and enhance the predominant static analysis methods.
- The design community and synthesis tool providers shall converge to a standard synthesizable subset. On top of the standard, industry and academia shall collaborate to make available a set of reusable templates and libraries as references for efficient synthesis of common design patterns. The reference templates and libraries should be relatively efficient in execution time and memory footprint.
- The synthesizable reference model should contain as few microachitecture and implementation details as possible. Too much target-dependent details would slow down the simulation and compromise the retargetability of the ESL model. This requires the synthesis tools support the complete separation of the target platform description and the synthesis constraints from the source code. In addition, many common code transformations (such as unrolling, inlining, streaming, etc.) should be applied automatically based on the given performance/area/power constraints. The out-of-box QoR[2] is becoming increasingly important to minimize manual changes to convert unstructured software code into synthesis-friendly specifications.

[2] Out-of-box QoR means the QoR produced by the tool automatically without any or much of the user intervention via pragmas, directives, or additional tool-specific (not algorithm-specific) code changes.

We would like to mention that although retargetability is important for modeling in the sense that the functional design can be easily remapped to a different technology or a different device, high-level synthesis and optimization tasks have to consider specific architecture characteristics and constraints in order to achieve high quality of results. We introduce these details next.

3. Directions in Synthesis and Optimization

In this section we address three important HLS topics and discuss the challenges and opportunities in each area. Specifically, Section 3.1 explains the importance of advanced synthesis for both on-chip and off-chip memories; Section 3.2 stresses the need for effective low-power analysis and optimization at the high level; Section 3.3 suggests new research directions on variation-aware high-level synthesis.

3.1 Advanced Memory Synthesis

3.1.1 On-Chip Memories

Modern SoC designs use over 50% of the area on embedded memories that serve as FIFOs, line buffers, look-up tables, scratch-pads, and caches to store data for the microprocessors and computational accelerators. These on-chip memories also contribute 50-70% of the total power dissipation. Undoubtedly, the choice of memory architecture at the high level is critical to the quality of the final design.

Interestingly, the majority of HLS research has been primarily focusing on reducing the schedule latency under resource constraints (in terms of functional unit/register count) or vice versa, with recent studies taking reliability, temperature and yield into further consideration. The memory optimization, however, is often an afterthought. As indicated by STMicroelectronics in [8], memory accesses appear to them as the most limiting factor for HLS exploration and optimization. Many existing solutions rely on simple method to create memory blocks based on bit width and size of data arrays specified in the source code. As a result, the limited memory ports often become the performance bottleneck. Furthermore, using oversized memory blocks would create wiring detours and routability problem.

Recently, an automatic memory partitioning technique is proposed in [7]. It honors the given throughput constraints and analyzes the data access patterns to derive the best possible memory partitioning for high performance as well as low power. Other promising research topics along the same direction include automatic memory merging, reshaping, data reordering, etc.

3.1.2 Off-Chip Memories

The intelligent synthesis support of external off-chip memories is equally important, especially with the high-definition trend and the rise of reconfigurable computing:

- Highly data-intensive video/image processing applications often require multiple frames of data be stored on DDR SDRAMs. Fast and efficient direct memory access logic needs to be in place to achieve high performance.
- Recent advances in FPGA-based high-performance reconfigurable computing [10] also require efficient access

to the gigabytes external memories shared by the host processor and the FPGA accelerator.

As mentioned in [18], most of the existing C-to-gates solutions currently lack efficient support of the memory hierarchy and sufficient abstraction of the external memory accesses. As a result, the programmers are exposed to the low-level details of bus interfaces and memory controllers. They must be familiar with the bus bandwidth and burst length and translate such knowledge to C code with substantial modifications. Clearly, such design practice is out of the comfort zone for many software developers and algorithm designers.

Hence it is highly preferable to have synthesis tools hide explicit external memory transfers as much as possible from programmers. This would require the support of efficient memory hierarchies including automatic caching and prefetching. The CHiMPS project [20] is one of the promising attempts in this area. The proposed C-to-FPGA compilation flow generates multiple distributed caches used by multiple concurrent processing elements. The compiler utilizes dependence and alias analysis to determine data clustering for higher degree of parallelization.

3.2 Effective Power Analysis and Optimization

Low-power design requires users assess and optimize the system architecture as early as possible in the design flow [19]. Trying to optimize for low power at RT level is important but likely has much less impact than high-level decisions such as hardware/software partitioning, bus width sizing, pipelining, etc. However, estimating power accurately at the high level remains to be very challenging. A great deal of low-level implementation details need to be considered to estimate the power consumption in a relatively accurate manner. For instance:

- Sophisticated activity propagation across registers and internal signals is required to obtain high correlation with the real circuit switching behavior.
- Clock tree modeling is critical to capture the clock power. The impact of extensive clock gating must be taken into account as well.
- Physical prototyping is needed to bypass the actual implementation and estimate the silicon area and interconnect power.
- The increasing usage of multi-voltage islands, dynamic voltage frequency scaling and power gating add further complexities to the power equation.

Currently, most designers (esp. in ASIC world) are still relying on time-consuming gate-level power analysis flows to obtain highly accurate power. It typically takes days to measure the power consumption in the netlist back-annotated with VCD waveforms and SPEF files (after parasitic extractors).

Needless to say, this is a great time for research and development on high-level and system-level power analysis. Advances in this area have the potential to significantly reduce the turnaround time in achieving the power closure.

The technique proposed in [3] is one of the promising attempts along this direction targeting FPGA architectures. As a case study, we introduce this work in detail next.

Traditionally, people are more concerned with area and power of functional units and registers. As technology advances, the area and power of multiplexers and interconnects have by far outweighed the area and power of functional units and registers especially for FPGA architectures. We carried out high-level power analysis and optimization targeting FPGA chips in [3]. We concentrated on resource allocation and binding tasks because they are the key steps to determine the interconnections during high-level synthesis. To fully validate our methodology and result, we target a real FPGA architecture — Altera Stratix architecture, which includes generic logic elements, DSP cores, and different types of memories, etc. We design a high-level power estimator for this architecture and verify that its power estimation result is very close to that reported by Altera's gate-level power estimator — Quartus II PowerPlay Analyzer. We form, propagate and prune binding/allocation solution points guided by our power and delay estimation. During this process, we pay attention to interconnects and multiplexers to control their power consumption and delay. Eventually, we generate a design solution curve, which can provide ideal solution points with low power and high performance.

Table 1: Area estimation functions for common operations on Altera Stratix FPGAs (*N*: bitwidth; *K*: # of input operands) [3].

Operation	Resource	Usage
Add/Subtract	LE	N
Bitwise and/or/xor	LE	N
Compare ($=, >, \geq$)	LE	$round(0.67*N+0.62)$
Shift (with variable shift distance)	LE	$Round(0.045*N^2+3.76*N-8.22)$
Multiply	DSP9x9	$N \leq 18$: $\lceil N/9 \rceil$ $N \leq 36$: $\lceil N/18 \rceil$
Multiplexer	LE	$N*round(0.67*K)$

Since we target Stratix FPGA device families, we need to deal with Stratix-specific features in our high-level power estimation. The area estimation functions for the multiplexers and several commonly occurring arithmetic units are listed in Table 1. Note that due to the high regularity of the FPGA device, the final resource usages of most operations are very predictable and their estimation functions can be expressed in close-form equations.

For switching activity calculation, we extend a method published in [1], which performs simulation just once at the beginning at the behavior level and computes switching activities for any legal binding without repeating simulations afterwards. We add loop support in the method. Based on this area and switching activity estimation, dynamic power of various types of resources can be estimated which will guide the low power optimization procedure. We also model the delay characteristics for these resources. We then are able to construct path delays by modeling the details of the datapaths during the design space exploration process.

Overall, our high-level power estimator is only 8.7% away from a commercial gate-level FPGA power estimator. Comparing to a traditional graph coloring-based register binding algorithm, our algorithm is 32% better on power and 16% better on Fmax after placement and routing. This demonstrates the effectiveness of HLS for power reduction and performance improvement. It also shows that power/delay modeling needs to pay attention to architecture-specific features. Good modeling strategies can offer

great accuracy without paying a large penalty in terms of runtime.

In another work, we performed HLS targeting glitch power reduction [9]. Glitches (i.e. spurious signal transitions) are major sources of dynamic power consumption. We target FPGAs in this study as well. Our binding algorithm employs a glitch-aware dynamic power estimation technique derived from the FPGA technology mapper in [4], which is utilized to reach to lower-level implementations. We design a cost function that is able to optimize both switching activity and path balancing to reduce glitches. High-level binding results are converted to VHDL, and synthesized with Altera Quartus II software, targeting the Cyclone II FPGA architecture. Power characteristics are evaluated with the Altera Power-Play Power Analyzer. The results of our algorithm are compared to LOPASS [2], a state-of-the-art low-power high level synthesis algorithm for FPGAs. Experimental results show that our algorithm, on average, reduces toggle rate by 22% and area by 9%, resulting in a decrease in dynamic power consumption of 19%. Figure 2 shows the details. $\alpha = 0.5$ indicates that the algorithm optimizes both switching activities and path balancing (through balancing multiplexers in the datapath), where $\alpha = 1$ only optimizes switching activity. We observe that $\alpha = 0.5$ produces reductions for all benchmarks, averaging 21.9%.

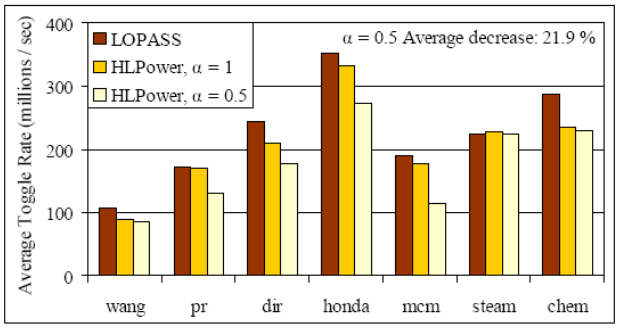

Figure 2: Average toggle rate reduction in [9].

It is worth noting that many powerful high-level low-power optimizations can be applied very effectively without the need of the most accurate estimation. For example, the concept of observability don't-care (ODC) is recently generalized at the behavior level in [5] to guide the compilation and synthesis to identify and avoid (via gating or shutdown) unnecessary computations, memory accesses and data transfers[3]. RTL-based analysis can hardly derive such high-level ODC information completely and efficiently. Moreover, behavior-level ODC can be more powerful when combined with HLS optimization.

A simple example is shown in Figure 3 to demonstrate the impact on scheduling on clock gating. In Figure 3 (a), the comparison operator is performed after the multiplication and none of the registers can be clock gated. In Figure 3 (b), the comparison is scheduled before the multiplication. As a result, when the comparison result is false and only value '*A*' is observable after the multiplexer selection, two pipeline registers can be gated and the activity of the multiplier can be saved.

[3] Such high-level ODC information can hardly be derived from RTL in an efficient and complete manner.

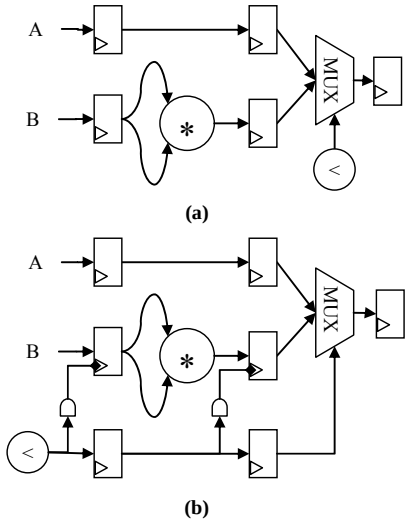

Figure 3: Scheduling impact on clock gating.

In [6], behavior-level ODC is considered in an intelligent soft-constrained scheduling algorithm. Soft constraints are actually preferences rather than essential constraints (referred to as hard constraints). Unlike hard constraints, soft constraints are supposed to be followed whenever possible but not necessarily. Specifying soft constraints would offer better design space characterization. Experiments show that an average of 33.9% reduction in total power can be achieved with close-to-optimal solutions on many real-life designs.

3.3 Variation-Aware High-Level Synthesis

Aggressive technology scaling to the deep sub-micron realm has resulted in significant variations in fabricated device parameters. In turn, these parameter variations have caused many traditional circuit design and analysis techniques to become inadequate. To overcome this obstacle, a shift in the design paradigm from the worst-case deterministic design to a statistical or probabilistic design is critical.

A new era of statistical design techniques has begun to emerge where circuit parameters such as delay and power are no longer modeled as deterministic values, but are represented as probability density functions. These statistical design techniques are leading to reclamation of lost performance and yield that has been occurring when using deterministic design techniques.

The shift to probabilistic design methodologies has produced a number of gate-level variation-aware optimization techniques [12][15]. While progress at the gate-level is encouraging, the large productivity gains available in high-level synthesis make it attractive and necessary to address the issue of process variations at a higher level of abstraction. Reference [13] offers a simultaneous scheduling, binding, and allocation algorithm based on simulated annealing. The simulated annealing algorithm seeks to reduce the overall latency while meeting a performance yield requirement. However, the algorithm does not consider multiplexer use or interconnect delay, both of which can significantly contribute to the clock period of the unit. Reference [14] proposes a timing variation-aware HLS algorithm which improves resource sharing. While the algorithm is effective, it ignores multiplexers and interconnects, and also

relies on the assumption that functional units are independent of each other in its yield calculation.

Recently, we proposed a novel variation-aware simultaneous binding and module selection algorithm, named FastYield, which maximizes the performance yield of the resulting circuit. We connect our synthesis engine to the layout closely, so layout information can be accurately back-annotated to the synthesis and introduce useful synthesis transformations. Synthesis and layout are iterated until the performance gain is maximized. We consider registers, multiplexers, functional units, interconnects, and spatially correlated process variations. A timing-driven, simulated annealing-based, statistical floorplanner that considers interconnect delay and spatial correlation between all units in the design is used to provide layout information to feedback to the HLS engine. On average, FastYield achieves an 85% performance yield clock period that is 14.5% smaller, and a performance yield gain of 78.9%, when compared to a variation-unaware and layout-unaware algorithm. This result shows that by performing statistical layout-driven synthesis, substantial gains in performance yield can be made during HLS.

Future work along this direction includes making scheduling variation-aware as well. Simultaneous register and functional unit binding considering process variation will also be considered. Challenges remain for controlling the overall runtime and avoiding conflicts between HLS optimization and layout optimization.

4. Concluding Remarks

This paper identified a set of critical needs and key challenges in ESL design automation with special focus on high-level synthesis (HLS) [4]. Specifically, we discussed the hardware synthesis challenges implied by software-centric ESL models and those required for optimizations of memory hierarchy, low power and process variations. These needs and challenges have created many new and important research directions as well as business opportunities in the EDA community. We hope that this paper would help to stimulate more research and activities in these areas.

REFERENCES

[1] A. Bogliolo, et. al, "Efficient Switching Activity Computation During High-Level Synthesis of Control-Dominated Designs," *Intl. Symposium on Low Power Electronics and Design*, Aug. 1999.

[2] D. Chen, J. Cong, Y. Fan, and L. Wan, "LOPASS: A Low-Power Architectural Synthesis System for FPGAs with Interconnect Estimation and Optimization," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, to appear.

[3] D. Chen, J. Cong, Y. Fan and Z. Zhang, "High-Level Power Estimation and Low-Power Design Space Exploration for FPGAs," *IEEE/ACM Asia and South Pacific Design Automation Conference*, Jan. 2007.

[4] Note that we haven't covered several other important topics such as verification and IP integration for ESL in this article.

[4] L. Cheng, D. Chen, and D.F. Wong, "GlitchMap: An FPGA Technology Mapper for Low Power Considering Glitches," *IEEE/ACM Design Automation Conference*, Jun. 2007.

[5] J. Cong, B. Liu, and Z. Zhang, "Behavior-Level Observability Don't-Cares and Application to Low-Power Behavioral Synthesis," *Intl. Symposium on Low Power Electronics & Design*, pp. 139-144, August 2009.

[6] J. Cong, B. Liu, and Z. Zhang, "Scheduling with Soft Constraints," *Intl. Conference on CAD*, November 2009.

[7] J. Cong, W. Jiang, B. Liu, and Y. Zou, "Automatic Memory Partitioning and Scheduling for Throughput and Power Optimization," *Intl. Conference on CAD*, November 2009.

[8] P. Coussy and A. Morawiec, eds., "High-Level Synthesis: From Algorithm to Digital Circuit," Springer, 2008.

[9] S. Cromar, J. Lee, and D. Chen, "FPGA-Targeted High-Level Binding Algorithm for Power and Area Reduction with Glitch-Estimation," *IEEE/ACM Design Automation Conference*, July 2009.

[10] T. El-Ghazawi, E. El-Araby, M. Huang, K. Gaj, V. Kindratenko, and D. Buell, "The Promise of High-Performance Reconfigurable Computing," *IEEE Computer*, vol. 41(2), pp. 69-76, Feb. 2008.

[11] F. Ghenassia, "Transaction Level Modeling with SystemC," Springer, 2005.

[12] M. Guthaus, N. Venkateswaran, C. Visweswariah, V. Zolotov, "Gate Sizing Using Incremental Parameterized Statistical Timing Analysis," *Intl. Conference on CAD*, 2005.

[13] W. L. Hung, X. Wu, and Y. Xie, "Guaranteeing Performance Yield in High-Level Synthesis," *Intl. Conference on CAD*, 2006.

[14] J. Jung and T. Kim, "Timing Variation-Aware High Level Synthesis," *Intl. Conference on CAD*, 2007.

[15] I. Lin, T. Ling, Y. Chang, "Statistical Circuit Optimization Considering Device and Interconnect Process Variations," *Intl. Workshop on System Level Interconnect Prediction*, 2007.

[16] M. Moy, F. Maraninchi, and L. Maillet-Contoz, "Pinapa: An Extraction Tool for SystemC Descriptions of Systems-On-a-Chip," *ACM Intl. Conference on Embedded Software*, pp. 317-324, September 2005.

[17] G. Martin and G. Smith, "High-Level Synthesis: Past, Present, and Future," *IEEE Design & Test of Computers*, vol. 26(4), pp. 18-25, December 2009.

[18] S. Neuendorffer and K. Vissers, "Streaming systems in FPGAS," in SAMOS Workshop, ser. *Lecture Notes in Computer Science*, no. 5114, pp. 147-156, July 2008.

[19] M. Pedram, "Low Power Design Methodologies and Techniques: An Overview," http://atrak.usc.edu/m. assoud/Papers/LPD-talk.ps, 1999.

[20] A. Putnam, S. Eggers, D. Bennett, E. Dellinger, J. Mason, H. Styles, P. Sundararajan, and R. Wittig, "Performance and Power of Cache-Based Reconfigurable Computing," *Intl. Symposium on Computer Architecture*, June 2009.