Practical Aspects of HLS Tools

High-Level Synthesis for Every Day Electronic Engineering and IC Design

Michael Dossis
Dept. of Informatics Engineering
TEI of Western Macedonia
Kastoria, Greece
dossis@kastoria.teikoz.gr

Abstract— **The current complexity of custom and embedded IP cores and integrated electronics requires a new generation of automated system design and development methods. High-Level Synthesis plays a critical part towards this direction and a number of research HLS projects are active. Nevertheless, existing HLS tools are not widely accepted by the engineering community for a number of practical reasons. This article is a practical perspective of HLS, and it includes an extensive discussion of related practical issues. Furthermore, this work includes a useful introduction to the system engineer that wants to consider HLS as part of his everyday system design practice. An alternative HLS methodology is presented that the author has developed and which is based on formal methods, thus it guarantees the correctness of the synthesized hardware and system. The paper concludes with an extensive experimental evaluation and it draws a number of suggestions about the future directions of HLS technology which is actually needed by the engineering community.**

Keywords- High-Level Synthesis; VLSI IC design; ESL tools; Electronic Design Automation

I. Introduction

Digital microelectronics found in embedded, high-performance and portable computing systems have highly complex components, design hierarchy and interconnections. This design complexity cannot be dealt anymore with conventional methods such as RTL coding, which suffer from prolonged development times, due to heavily slow and iterative design flows, which often causes products to miss the market windows. During the last couple of decades, commercial and academic organizations have invested in High-Level Synthesis (HLS) and optimization techniques, so as to achieve design automation, quality of implementations and short specification-to-product times [1], [2].

Existing HLS tools have not yet gained wide confidence by the engineering community for a number of practical reasons. Such reasons include the sensitivity to the coding style of the input specifications as well as to the utilized HLS algorithmic heuristics, and the lack of support to a number of standard programming constructs (e.g. while loops) and programming language styles. Moreover, due to bugs that are introduced during the synthesis process, the engineer is burdened with the need for re-verifying the generated RTL hardware code to removes such errors.

The main contribution of this work is a discussion of practical issues of HLS tools and the proposition of such a tool developed by the author of this paper, which is based on formal methods. The formal transformations of the C-Cubed tools make the generated RTL hardware implementations provably-correct with respect to the input code. Moreover, the CCC compiler accepts all of the input programming styles, hierarchy, complex control and standard programming constructs such as nested if-then-else, for loop, while loop etc.

The next section includes a discussion of the existing academic and commercial HLS technology and research work. Section III includes a short discussion of the need for formal synthesis transformations. Section IV describes the C-Cubed EDA technology and the associated HLS and verification flows. Section V draws conclusions and a tool evaluation from a number of benchmark and real-work application experiments. Section VI includes a discussion about the required future directions of HLS technology and its practical impact to the engineering community. The last section draws useful conclusions and proposes future work on the C-Cubed EDA technology.

II. Practical Issues of HLS Tools

Research in High-Level Synthesis started in the 80s and the first robust linear processing HLS tools appeared in the academic and industrial labs, in the early 90s. Important problems that researchers of HLS were called to handle included the allocation, scheduling and binding problems. The most difficult of these three tasks is the building of a reliable scheduler [3]. It is well known that when the system complexity increases linearly, the complexity of the scheduler algorithm increases exponentially and for some applications, scheduling is NP-complete. This problem becomes even more critical and difficult in practice when input code with complex module and control flow hierarchy (e.g. nested while and for loops) is to be processed by the HLS tool [4], [7].

Existing HLS tools are still not widely accepted by the engineering community because of their poor results, especially for large applications with complex module and control-flow hierarchy. Very often, the programming style of the source code has a severe impact on the quality of the synthesized implementation. For large-scale applications, the complexity of the synthesis transformations (front-end compilation, algorithmic transformations, optimizing scheduling, allocation and binding), increases exponentially, when the design size increases linearly [3], [4], [5], leading to suboptimal solutions when synthesis heuristics are employed to cut down the long processing times.

Many existing HLS tools impose proprietary extensions or restrictions (e.g. exclusion of while loops) on the programming model of the specifications that they accept as input, and various heuristics on the HLS transformations that they utilize (e.g. guards, speculation, loop shifting, trailblazing) [2]. Most of them are suitable for only linear, and dataflow dominated (e.g. stream-based) designs, such as pipelined DSP, image processing and video/sound streaming.

The most important commercial existing HLS tools include the Catapult-C from Calypto (previously developed by Mentor Graphics), and Cynthesizer from Forte Design Systems. They both accept as input a small subset of System-C and C++. Both of these tools are too difficult to use by the average system developer and they are the most expensive of their class since they are licensed for something less than 300K dollars per year. Therefore, these E-CAD tools are very difficult to access for many small ASIC/FPGA design SMEs.

Other commercial or industrial HLS tools are the Symfony C compiler from Synopsys, the Impulse-C from Impulse Accelerated Technologies, the CyberWorkBench from NEC, the C-to-silicon from Cadence, and the free web-based tool C-to-verilog from an Israel-based group. Most of these tools are either used internally by the developing company, or they are not well-established amongst the engineering community for reasons that were explained above.

Amongst the academic or research-based HLS tools are the SPARK tool [2] which accepts as input a small subset of the ANSI-C language (e.g. while loops are not accepted), and a conditional guard based optimization method [7] which set the basis for processing conditional code in the beginning of the previous decade.

III. The Need for Formal Synthesis Methods

It concludes that what is needed from a HLS toolset is the incorporation of intelligent and formal techniques in order to apply the source-to-implementation optimizing transformations, and thus turn the produced hardware implementations to correct-by-construction. In this way, only top behavioral level verification (e.g. with rapid compile and execute of the specs) is required, without spending weeks or even months, on lengthy RTL or annotated gate simulations. Constraints and other options can be applied by the user on the automatic HLS transformation, such as the number of available resources, the length of the desired schedule, the type of the micro-architecture, the generated HDL code as well as the inclusion of custom (e.g. arithmetic) logic functions throughout the HLS compilation.

IV. The C-Cubed ESL HLS Toolset

The author has designed and developed an intelligent HLS compiler [4] that includes a scheduler of operations into control steps, achieving the maximum functional parallelism in the synthesized implementation [5]. It employs an advanced HLS scheduler called PARCS, which utilizes formal techniques such as logic programming [6] and RDF subject-predicate-object relations [7], to formally achieve the maximum possible parallelism of operations. In this way, the functionality of the delivered implementations is correct-by-construction.

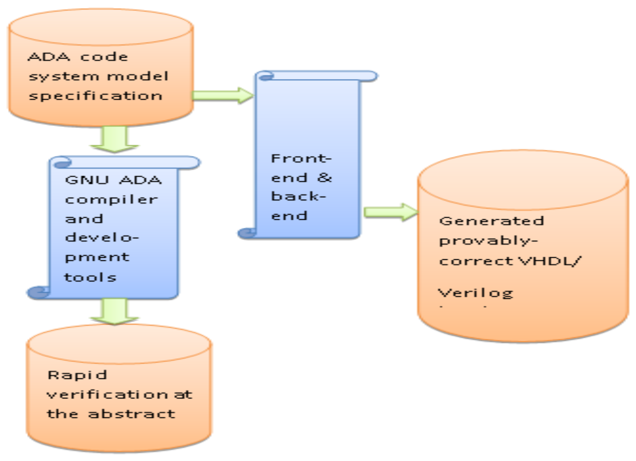
ADA code system model specification
Front-end & back-end
GNU ADA compiler and develo-pment tools
Generated provably-correct VHDL/ Verilog
Rapid verification at the abstract

Figure 1. The C-Cubed EDA Framework design flow

A detailed description of the above intelligent approach of the prototype optimizing CCC synthesizer can be found in [4]. The CCC tool employs advanced techniques such as formal predicate logic [6], RDF relations and XML schema validation to improve the quality of the synthesis results. The usability and correctness of the C-Cubed HLS toolset were evaluated with a large number of benchmarks. The CCC design flow is shown in Fig. 1.

The C-cubed ADA HLS design and verification flow, includes the front-end and back-end HLS tools, and the GNU ADA integrated compiler, development and verification environment. The full standard programming construct set of the ADA and ANSI-C language sets are accepted by the CCC synthesizer. The front-end compiler is a compiler-generator parsing and syntax processing system with all the standard software compiler optimizations. The back-end compiler is based on logic programming inference engine rules and it includes the formal PARCS scheduler and optimizer. PARCS attempts always to parallelize as many as possible operations in the same control step, as far as there are no dependency violations. However, the tool can be driven by external module and operator specific resource constraints.

V. Experiments, Tests and Tool Evaluation

Arbitrary and general input ADA or ANSI-C code is synthesized into functionally-equivalent RTL VHDL/Verilog hardware implementation. Many applications were synthesized with the C-Cubed toolset and they are extensively discussed in [4]. In any case, the functionality of the produced hardware accelerators (coprocessors) matched that of the input subprograms. This was expected due to the formal nature of the utilized HLS transformations and optimizations. More tests and benchmarks such as real-world image and telecom processing applications are under way and they will be published in future papers of the related research work.

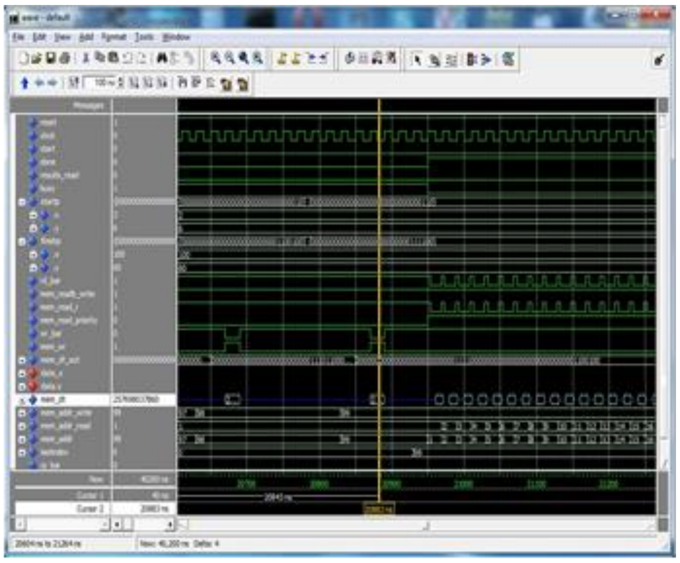

Figure 2. The synthesized computer graphics test simulation

After the tests were coded and verified in ADA they were synthesized into VHDL/Verilog RTL. Since the C-cubed tools are based on formal techniques there is no need to simulate the generated RTL. Nevertheless for proving this argument in practice we have simulated all the generated RTL tests to ensure that they feature an equivalent to that of the source code behaviour. A RTL simulation of a computer graphics benchmark generated HDL code is shown in Fig. 2. It is shown clearly in this figure that the generated hardware FSM completes its function with the synchronized done/results_read signal event, as well as all the external memory transactions after the completion point. It is clear that these transactions write the coprocessor's results into the external memory.

The following table shows the state reduction, using the PARCS optimizer for two benchmarks, the computer graphics line drawing algorithm and the MPEG engine. It is important to mention that in some cases of complex control flow, the state reduction rate reaches up to 41 per cent.

TABLE I. STATE OPTIMISATION WITH THE C-CUBED PARCS OPTIMIZER

Module name	***Initial schedule states***	***PARCS parallel states***	***State reduction***
line-drawing design	17	10	41%
MPEG 1st routine	88	56	36%
MPEG 2nd routine	88	56	36%
MPEG 3rd routine	37	25	32%
MPEG top routine (with embedded memory)	326	223	32%
MPEG top routine (with external memory)	462	343	26%

Many benchmarks and tests were synthesized with CCC tools. Some of them are: a DSP FIR filter, an MPEG engine, and a cryptographic RSA processor. The state reduction for these and other benchmarks (not mentioned here) is shown graphically in Fig. 3. All the tests were compiled with CCC in less than 10 minutes.

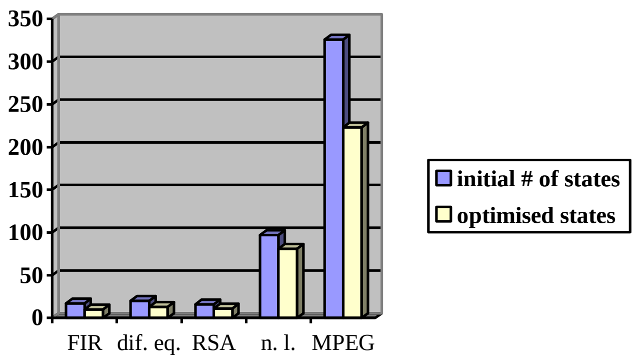

Figure 3. Graphical view of state reduction for the five benchmarks

The MPEG engine comprises of a FSM with more than 400 states! Such designs are practically impossible to design and verify manually and directly in RTL. Therefore the contribution of the C-cubed technology to the engineering community and industry is invaluable, in order to automate the hardware development and generation process.

VI. HLS PROSPECTS AND THE FUTURE

What about the future? What are current and future directions of industrial developments in HLS? More input programming languages (e.g. C++, System-C, UML, Fortran, Delphi-Pascal, Java) and a more globalized use of formal techniques throughout the flow of the HLS toolset are needed in order to bring wide acceptance of practical HLS results into the industry. Moreover, HLS methodologies need to be more adaptable to the needs of different engineering environments and many already established industrial backend flows.

Another important role of HLS in the engineering practice is the re-use of existing hardware and software IP. To achieve this a wide compatibility of HLS input/output with languages and formats is required to use HLS in practical every-day system engineering, as well as rapid prototyping capability to the future electronics product development. In order to achieve this, formal HLS transformations need to be implemented, which are not sensitive to heuristics and input code style. Moreover, arbitrary and complex module and control flow in the designer's set of system models need to be transformed with ease, speed and quality into the required software and hardware implementations.

VII. CONCLUSSIONS AND FUTURE WORK

Sometimes the assumptions that many existing HLS tools make about targeted technology attributes such as timing and power consumption, produce disappointing synthesis results, since there is still no established methodology for feeding target technology characteristics back into the core of the HLS transformation process (although some academic attempts to model this problem have been made). In many cases these target implementation characteristics need to be fed into the synthesis flow and guide the complex synthesis transformations of the HLS tool.

The C-Cubed synthesizer is making an important step towards the above directions and a number of related projects are under-way to deliver better synthesis results with readable RTL code and better visibility of the design's attributes and algorithmic features. Of course it is not the only attempt to deal with the complexities of the HLS transformations. There have been a number of research projects that target a better engineering environment to alleviate the frustrations of industries about dealing with development results that are just too late to hit the market window for many electronics products. The developments on the CCC technology, help to push forward with EDA tools with practical as well as theoretical value, since they are suitable to synthesize any style of input (specification) coded algorithms into functionally-equivalent hardware. All the standard programming language constructs are accepted without any exceptions or compromises, and more languages will be accepted as input and produced as output of the CCC tools in the future.

Future work for the C-cubed tools include the inclusion of a number of input language formats such as ANSI-C, C++, SystemC and OpenCL, and a number of output formats for quick verification like SystemC and cycle-accurate C. Also, a number of source code optimizations such as dynamic loop-unrolling and code motion are under study, experimentation and development. The capability of the re-use of synthesized IP as custom block libraries to the tool is investigated and related development work with complex arithmetic custom blocks and coding of multi-cycle and pipelined logic is expected to produce results soon.

Acknowledgment (Heading 5)

The presented work is partially supported by the TEI of Western Macedonia, dept. of Informatics Engineering, Kastoria, Greece.

References

[1] Gal, B. L., Casseau, E., and Huet, S. "Dynamic Memory Access Management for High-Performance DSP Applications Using High-Level Synthesis". IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 16, no. 11, pp. 1454-1464, 2008.

[2] Gupta, Sumit, Gupta, Rajesh Kumar, Dutt, Nikil D., and Nikolau, Alexandru. "Coordinated Parallelizing Compiler Optimizations and High-Level Synthesis". ACM Transactions on Design Automation of Electronic Systems, vol. 9, no. 4, pp. 441–470, 2004.

[3] Walker, R. A., and Chaudhuri, S. "Introduction to the scheduling problem", IEEE Design & Test of Computers, vol. 12, no. 2, pp. 60–69, 1995.

[4] Dossis, Michael F. "A Formal Design Framework to Generate Coprocessors with Implementation Options". International Journal of Research and Reviews in Computer Science (IJRRCS), August 2011, ISSN: 2079-2557, Science Academy Publisher, United Kingdom, www.sciacademypublisher.com, vol. 2, no. 4, pp. 929-936, 2011.

[5] Paulin, P. G., and Knight, J. P. "Force-directed scheduling for the behavioral synthesis of ASICs". IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 8, no. 6, pp. 661–679, December 1989.

[6] Nilsson, U., and Maluszynski, J. Logic Programming and Prolog. John Wiley & Sons Ltd., 2nd Edition, 1995.

[7] Kountouris, Apostolos A., and Wolinski, Christophe. "Efficient Scheduling of Conditional Behaviors for High-Level Synthesis". ACM Transactions on Design Automation of Electronic Systems, vol. 7, no. 3, pp. 380–412, 2002.