High-level Synthesis Integrated Verification

Michael F. Dossis
Dept. of Informatics Engineering
TEI of Western Macedonia
Kastoria, Greece
mdossis@yahoo.gr

***Abstract*—It is widely known in the engineering community that more than 60% of the IC design project time is spent on verification. For the very complex contemporary chips, this may prove prohibitive for the IC to arrive at the correct time in the market and therefore, valuable sales share may be lost by the developing industry. This problem is deteriorated by the fact that most of conventional verification flows are highly repetitive and a great proportion of the project time is spent on last-moment simulations. In this paper we present an integrated approach to rapid, high-level verification, exploiting the advantages of a formal High-level Synthesis tool, developed by the author. Verification in this work is supported at 3 levels: high-level program code, RTL simulation and rapid, generated C testbench execution. This paper is supported by strong experimental work with 3-4 popular design synthesis and verification that proves the principles of our methodology.**

Keywords-High-level Synthesis; Formal verification; E-DA

I. INTRODUCTION

Embedded, high-performance and portable computing systems digital circuits have highly complex design control & module hierarchy as well as interconnection schemes. This complexity cannot be dealt anymore with conventional methods such as Register Transfer Level (RTL) coding, since they involve highly iterative design flows, detailed and prolonged simulations and thus, prohibitive development times. These development problems often cause products to miss the market windows and thus, a lot of engineering effort is thrown in the rubbish bin. As a consequence, commercial and academic organizations have invested in High-Level Synthesis (HLS) and Electronic System Level (ESL) methodologies to achieve design automation, quality of implementations and short specification-to-product times [1-6]. Nevertheless existing HLS tools produce lower quality of implementations than manual methods, and the generated hardware models are difficult to handle. Also, during synthesis a great number of assumptions about the targeted computing platform and transformation heuristics are taken, in order to deal with the NP-complete synthesis problems, which makes synthesis results suboptimal [3, 5].

The most explored and formulated problems of HLS include high-level optimizations, scheduling, allocation and binding and they can be found in [1-6]. High-level optimizations are based on software compiler optimizations, allocation is selection of functional units and storing resources for the data and operations objects found in high-level program code, binding is the actual mapping of the above units to real hardware elements such as flip-flops, latches and combinatorial blocks such as functional operator hardware units, and scheduling is the arrangement of elementary operations to Finite State Machine (FSM) states in system's clock cycles. Although these problems have been studied in research labs, the optimization of real-world complex applications, as well as their mapping onto custom hardware fails to produce competitive to manual implementations. This is because they are not able to handle complex, nested control flow constructs and large data objects, as well as sophisticated interfaces through complex hierarchy and module configuration and to-from external shared memory interfaces.

Logic programming [7] and compiler generators found in other areas, such as artificial intelligence and software compilers can benefit synthesis tools. The Cubed-C framework [4] combines techniques from compiler generators and artificial intelligence logic programming so as to deliver formal synthesis of high-level programs (in ADA or C) into fully synthesizable and simulatable RTL (VHDL or Verilog) and executable testbenches (C). In this way specification to product development times are compressed significantly, since lengthy RTL or gate-level simulations are avoided.

II. RELATED WORK AND BACKGROUND

A. High-level Synthesis Tools

The first High-Level Synthesis research results emerged in the 80s and the first linear design (data-flow oriented) processing HLS tools appeared in the early 90s. The most complex of the synthesis tasks is the building of a reliable scheduler [3]. From relevant research it is evident that when the system complexity increases linearly, the complexity of the scheduler algorithm increases exponentially and for some very complex applications, scheduling is NP-complete [3, 6]. The difficulty to handle complex code control becomes critically difficult and it can prevent synthesis whatsoever in practice when source code models with complex module and control flow hierarchy (e.g. nested while, if-then and for loops) are encountered [4, 6].

Academic or commercial HLS tools are still not accepted by the engineering community because of the poor synthesis

results. This is true, particularly for large applications with complex module and control-flow hierarchy as stated above [3-5]. However, real-world applications can be benefited the most from automated techniques such as HLS. Often HLS tools produce suboptimal solutions when synthesis heuristics are used to shorten the long optimization and processing times.

Most of the existing HLS tools are suited for linear, and dataflow dominated (e.g. stream-based) applications, such as pipelined Digital Signal Processing (DSP), image processing and video/sound streaming. Nevertheless even such straightforward applications are not possible to process when the specification model contains any of the excluded programming constructs (such as while loops and loop breaks). Most of the 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]. However, when design complexity is over than a dozen states, manual RTL coding becomes prohibitive for delivering products in realistic times.

Some of the existing commercial HLS tools are 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 the System-C and C++ languages. However, both of these tools have too complicated user interfaces, and package libraries, for the average system modeler and developer and they are the most expensive of their class since they are licensed for something less than 300,000 dollars per year. In this way, commercial synthesizers are inaccessible for most of small and medium sized ASIC/FPGA (Application-Specific Integrated Circuit/Field Programmable Gate Arrays) design SMEs. Other HLS tools are Symfony C compiler from Synopsys, Impulse-C from Impulse Accelerated Technologies, CyberWorkBench from NEC, C-to-silicon from Cadence, and the free web-based tool C-to-verilog from an Israel-based group, all accepting small C subsets as input and mainly linear dataflow-oriented applications.

Popular academic 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 optimizer [6] forming the basis for optimizing conditional source code methods at the beginning of the previous decade.

A multi-speculative technique that composes complex adders during datapath synthesis, contributes only towards linear data flow oriented designs can be found in [8]. In [9] a fixed-point accuracy analysis and optimization of polynomial data-flow graphs is analyzed, with respect to a reference model that is found in many digital signal processing applications. A technique to improve nested loop pipelining for High-Level Synthesis, called Polyhedral Bubble Insertion is discussed in [10]. An equivalence checking method of FSMs with datapaths based on value propagation over model paths, for validation of code motion transformations usually applied during the HLS scheduling phase can be found in [11]. A formal HLS method for accurate high-level casting of optimal adders and subtractors is analyzed in [12]. An exploration approach, called spectral-aware Pareto iterative refinement, that uses response surface models (RSMs) and spectral analysis for predicting the quality of the design points without resorting to costly architectural synthesis procedures is presented in [13]. However, most of these approaches concern only either linear, dataflow oriented applications or attempt to solve only minor sub-problems of HLS, and therefore they don't impact the complete synthesis flow as a whole system.

B. Integrated Verification, the Cubed-C approach

In our approach, the high-level C or ADA model is first executed to confirm the correctness of the specification model in a rapid, compile-and-run way. When the specification executable model is fully debugged, it is passed to the Cubed-C HLS synthesizer to generate correct-by-construction, Register Transfer Level (RTL), VHDL/Verilog implementations. Computation-intensive algorithms and other real-world application can in this way be rapidly prototyped. RTL simulations are executed to verify the functionality and prove the correctness of the automatically generated RTL VHDL implementations, which is expected due to the formal nature of the synthesis transformations. The Cubed-C synthesis includes the frontend and the backend compilers, which communicate with each other via the ITF database. ITF is a formal Prolog-like format for capturing all of the semantic categories of a programmed algorithm. The frontend compiler is built using compiler-compiler techniques and the backend compiler using logic programming techniques [4]. Moreover the ITF syntax and semantics are formally defined in [14]. All of these methods are protected with international patents [15]. The PARCS scheduler is used in the backend runs to optimize the generated operations schedule. PARCS is a formal optimizer which parallelizes as many operations in the same clock cycle as possible, as long as control/data dependencies and resource constraints are obeyed, unless specific block/operator constraints are provided by the user.

The Cubed-C tools now automatically and formally generate ANSI-C testbenches that capture the functionality of the hardware FSM that is synthesized with the HLS tool. These testbenches are cycle-accurate simulators that can be compiled and executed with any of the available C development environments such as the GNU-C compiler and linker. Thus, the user can see through the FSM state transactions the changes in the values of registers. Moreover, the following options are possible: resetting the registers, providing input to all the inputs of the design and examining the values of the output registers of the design at any time in the cycle-accurate simulation. This method is formal but different than formal verification, since the Cubed-C method integrates both synthesis and verification methods in the same run. Therefore, functional and FSM bugs can be caught and corrected in an easy and a rapid manner.

This integrated synthesis and verification flow in the Cubed-C framework is shown in Figure 1, where red frames show verification tasks and blue frames show synthesis tasks. The high-level verification can be compared manually or automatically with a suitable testbench with cycle-accurate simulation (which was done in our experimental tests at the end of this paper) to confirm the matching of the two functionalities.

During recent years, model checking is a formal means of confirming the system functionality with mathematical assertions. However it is not practical for large applications where the state space is increasing to prohibiting levels. An attempt to deal with this is reported in [17]. A method based on a formal attribute grammar, used to translate human language into correct Computation Tree Logic from specific comments in the HDL code, is described in [18]. Transformations of the behavioral RTL FSMD model are proposed in [19] in order to deal with false negative results of the equivalence checking of state sequences from conditional paths in equivalent checking. Static and dynamic analysis of the RTL source code is exercised in [20] to define and compute code coverage of assertions in the RTL code. Simulation-based verification is executed in [21] with generated scenario-based stimuli which are triggered in various ways. The effort aims to compact the stimuli while satisfying the coverage requirements.

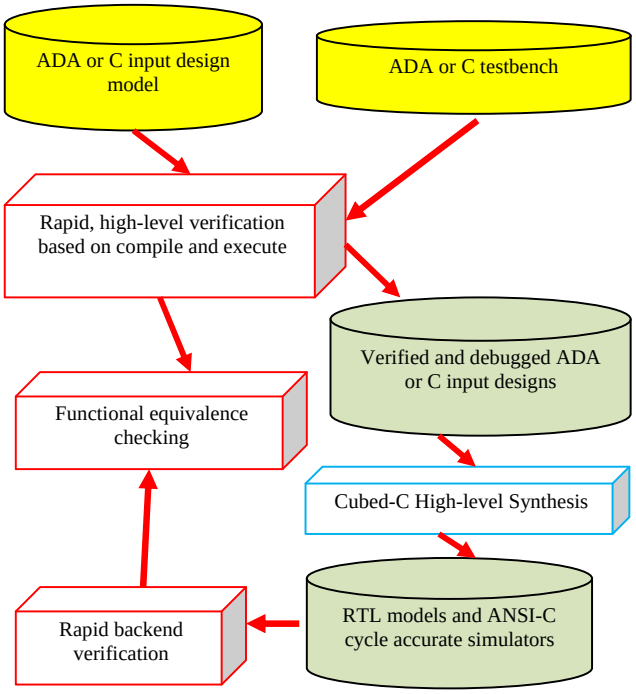

Fig. 1. Integrated, formal verification in the Cubed-C synthesis flow

III. INTEGRATED HLS VERIFICATION

The integrated with High-level Synthesis verification flow is depicted in Figure 1. The relevant tasks start with writing and debugging the C or ADA code which is the executable specification model at the highest possible abstraction level. This is done in a rapid manner since well understood compile and execute is performed on the improved versions of the high level model's code. Along with the specification code modules, special testbench modules are coded in C or ADA which help the functional debugging process to complete sooner. It must be noted that although the specification module should follow the Cubed-C coding guidelines, the testbench and any other modules built for debugging are free to code in any style of the ANSI-C or GNU-ADA programming languages.

After removing all the functional bugs quickly and rapidly at the highest functional level, then the debugged ADA/ C model code is passed to the Cubed-C synthesizer which automatically, rapidly and formally generates functionally-equivalent and provably-correct VHDL or Verilog RTL synthesizable models. These models are synthesizable with any commercial or academic RTL synthesizers into technology IC implementations. The Cubed-C compiler backend optimizes the state schedule into optimal FSMs and datapaths and generates from the same optimized FSM model a cycle-accurate ANSI-C compatible simulator for each of the ADA/C subprograms (design modules). The generated simulators can be rapidly compiled and executed to fully debug the generated modules in a much faster way than RTL, and without involving the designers with the hardware details of the RTL code.

Each of the generated cycle-accurate simulator steps includes options for resetting all registers, setting the inputs, advancing to the next FSM step (state) cycle, and reading the registers and the outputs of the circuit. All these are done with just pressing a keyboard key and quickly re-verify the provably-correct behavior of the generated circuit in a rapid manner. The user can write in any style the RTL testbench and the automated testbench to automatically or semi-manually compare the test vectors and outputs of the generated circuit against the vectors that were used in the abstract functional level, as clearly shown in Figure 1.

IV. VERIFICATION EXPERIMENTS AND RESULTS

There have been run many integrated synthesis and verification experiments through the design flow of Figure 1. Three of them will be discussed analytically here: A computer graphics algorithm, a DSP FIR (Finite Impulse Response) filter and the classical high-level synthesis benchmark, the second order differential equation approximation solver.

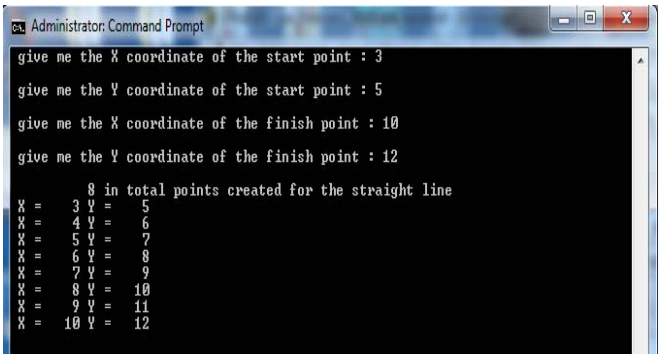

Fig. 2. The high-level ADA model verification

A. *The computer graphics algorithm*

The computer graphics application is a line drawing algorithm with the Digital Differential Analyzer (DDA) approximation method. Due to its iterative and conditional nature is a good representative for testing the Cubed-C's performance to the loop and if-then type of behaviors.

First the line drawing algorithm is coded in ADA (a similar experiment is done using the C language entry). Also, an ADA-based testbench is coded and the whole set of programs is compiled, executed to verify the correct behavior at this high level. Figure 2 shows the high-level testbench run, where the input start and end points as well as the intermediate line points are clearly formulated and shown in this experiment run.

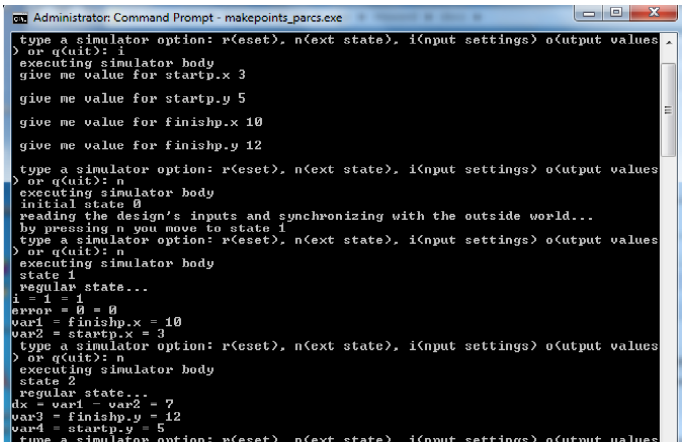

Fig. 3. Beginning of he cycle-accurate simulation of the line draw app

After the high-level model is debugged and verified it is synthesized into a RTL FSM and the cycle-accurate simulator is extracted from the machine. Figure 3 shows the beginning of the cycle-accurate simulation where the start and end points of the line are entered into the simulator. Then the simulator is executed cycle by cycle and this is shown in Figure 4.

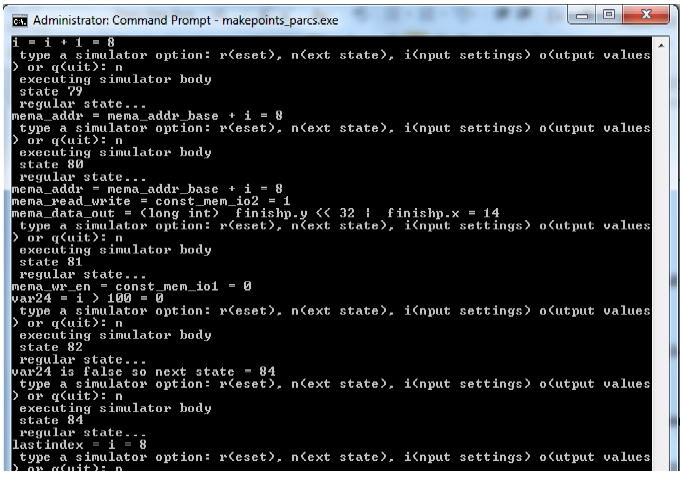

Fig. 4. Running the line draw cycle-accurate sim. cycle by cycle

After the last state is reached (which is indicated with a message to the user) the outputs of the simulator can be read as shown in Figure 5. Because we used the option to position of the line's points into an external shared memory we notice in Figure 5 the last transactions on the signals that interface with the memory modules. Alternatively we could follow the synthesis option to position the line's points in embedded memory in the line draw processor, but this exceeds the purpose of this paper.

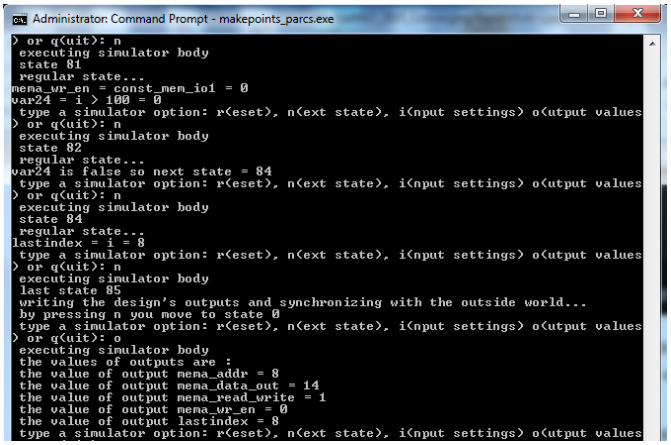

Fig. 5. Reading the simulator's outputs

B. Simulating the RTL for reassurance

In order to reassure in practice the formal nature of the synthesis process the generated RTL for the line draw app. is simulated as shown in Figure 6 and Figure 7. Figure 6 shows the whole duration of the generated RTL execution and Figure 7 the time duration of the transactions with the external memory of the FSM results. Figure 7 clearly shows the transfer of the 8 line points to the external memory.

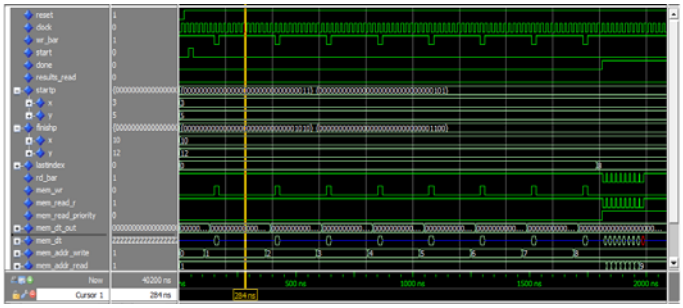

Fig. 6. RTL simulation of the generated line draw hardware

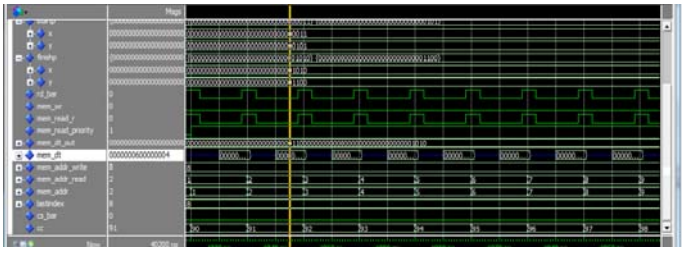

Fig. 7. Zoomed RTL simulation at the last 8 pixel transactions with the memory

C. Assertion-based testbench for line draw

Figure 8 shows the execution of an automatic, assertion based testbench that automatically compares the results (pixel by pixel) of the high-level and the RTL simulations. As expected there were no differences between the two. This testbench asserts suitable success/fail messages to the user, as it is shown in Figure 8. In this way, all 3 levels of integrated verifications can be co-verified and compared seamlessly.

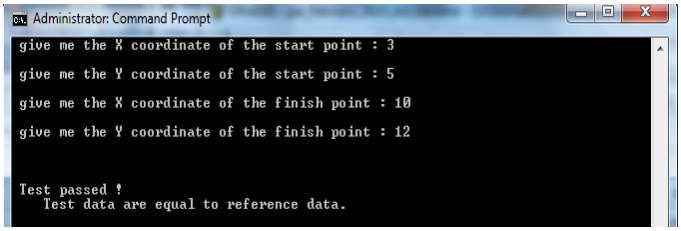

Fig. 8. Assertion-based, automatic testbench for the line draw

D. DSP FIR filter

The next benchmark to present here is a 5-tap DSP FIR filter model in ADA and subsequently in VHDL and cycle-accurate simulator runs. In this paragraph, the high-level and the cycle-accurate models execution is discussed, although RTL simulations were run as well. Figure 9 shows the execution of the algorithmic, high-level ADA model and testbench of the 5-tap filter. The setting of the history, the 5 coefficients and the outputs after 5 shift-outs are clearly shown in this picture. Figure 10 shows the Cubed-C backend cycle-accurate testbench execution during the beginning of the run, where the inputs, the history (taps) and the coefficients are set. Figure 11 shows the end of the cycle-accurate execution where the outputs are read and they are clearly the same as the high-level testbench of Figure 9.

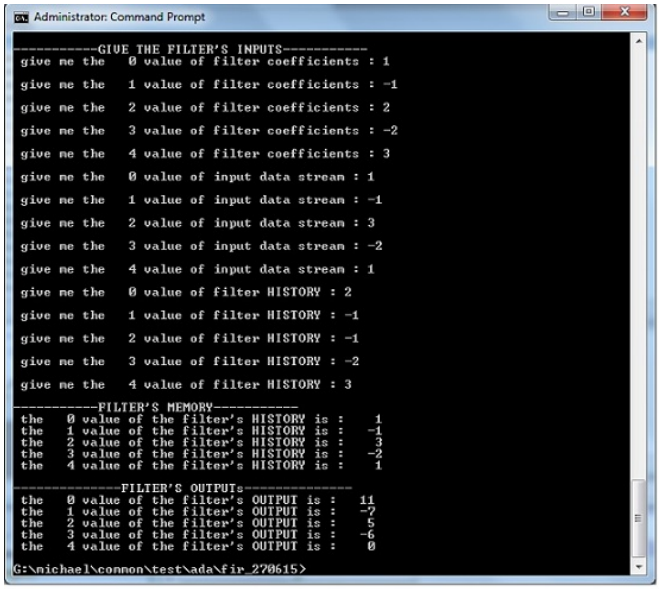

Fig. 9. Execution of the high-level ADA code model testbench

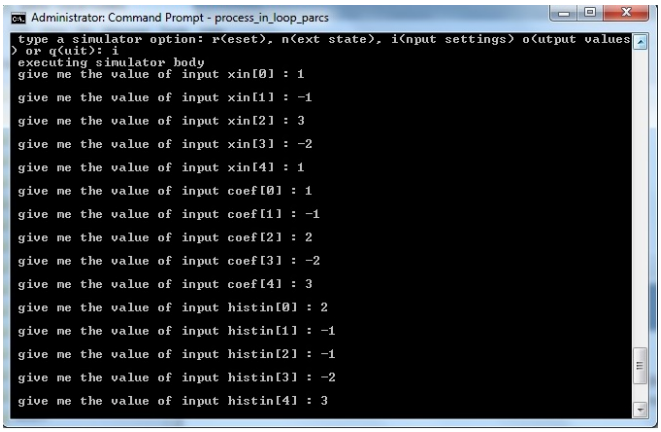

Fig. 10. Beginning of the FIR cycle-accurate simulation

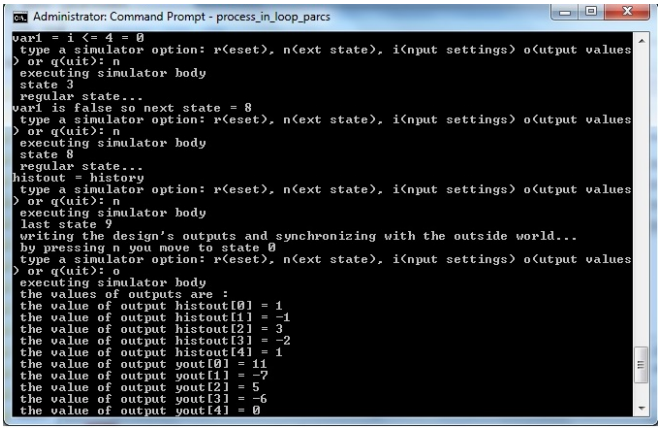

Fig. 11. Results of the completion of the cycle-accurate FIR simulator

E. The HLS differential equation solver benchmark

In this subsection the synthesis integrated verification experiment with the typical HLS benchmark is described, namely the second order differential equation solver. Figure 12 shows the execution of the functional, high-level ADA model and testbench execution in order to rapidly verify at this level the differential equation solver benchmark synthesis model. The same input values are used in the run of Figure 13 where the RTL simulation (giving the same result) of the synthesized circuit is shown. Figure 14 shows the cycle-accurate simulation near the beginning (setting the inputs) and Figure 15 shows the cycle-accurate simulation results from the same model that is produced by the same synthesis step as the RTL code. As expected in all cases, the results of all types of verification match.

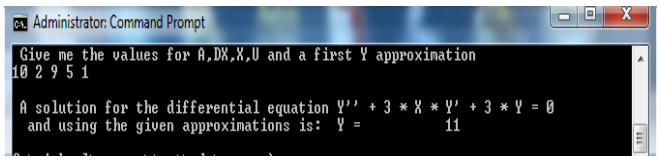

Fig. 12. Functional high-level testbench of diff. eq. solver

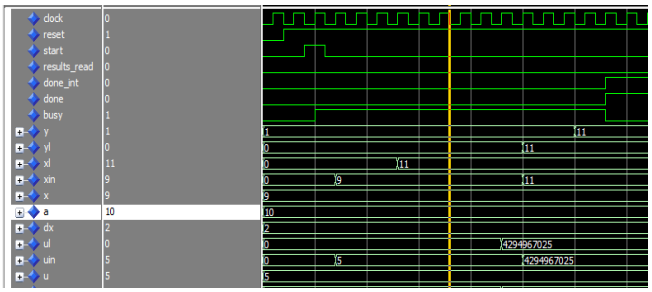

Fig. 13. RTL simulation of the synthesizeed diff. eq. solver circuit

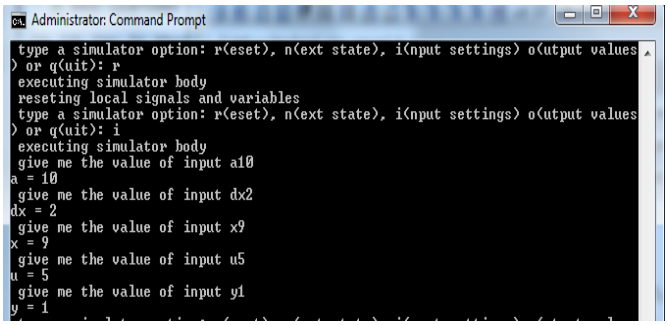

Fig. 14. Setting the inputs of the diff. eq. solver cycle-accurate model

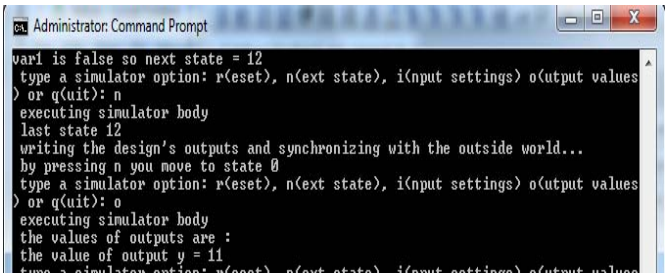

Fig. 15. Results of the cycle-accurate simulation

V. CONCLUSSIONS AND FUTURE WORK

A formal, rapid and integrated with synthesis, verification approach based on the Cubed-C High-level Synthesis system was discussed in this paper. The major contribution of this work is that the presented approach cuts down by orders of magnitude the verification effort required for digital design projects. All the presented simulations were completed in matters of seconds and the formal nature of the synthesis and testbench generation mechanism guarantees the correctness of the generated implementations. In this way the synthesized circuit features the same behavior as the input specification. Moreover, by having as specification executable code models makes the whole flow into a rapid, formal and seamless as well. Also, no modifications on the model and testbench code were required and raw programming language constructs were used. Future work includes generating cycle-accurate models in more verification languages such as C++ and System-C, and better use of the backend HDL writer for System Verilog flows.

REFERENCES

[1] B. L. Gal, E. Casseau, S. Huet, "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] S. Gupta, R. K. Gupta, N. D. Dutt, A. Nikolau, "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] R. A. Walker, S. Chaudhuri, "Introduction to the scheduling problem", IEEE Design & Test of Computers, Vol. 12, No. 2, pp. 60–69, 1995

[4] M. F. Dossis, "A Formal Design Framework to Generate Coprocessors with Implementation Options", International Journal of Research and Reviews in Computer Science, Vol. 2, No. 4, pp. 929-936, 2011

[5] P. G. Paulin, J. P. Knight, "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, 1989

[6] A. A. Kountouris, C. Wolinski, "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

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

[8] A. A. Del Barrio, R. Hermida, S. O. Memik, J. M. Mendias, M. C. Molina, "Multispeculative Addition Applied to Datapath Synthesis", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 31, No. 12, pp. 1817-1830, 2012

[9] O. Sarbishei, K. Radecka, "On the Fixed-Point Accuracy Analysis and Optimization of Polynomial Specifications", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 32, No. 6, pp. 831-844, 2013

[10] A. Morvan, S. Derrien, P. Quinton, "Polyhedral Bubble Insertion: A Method to Improve Nested Loop Pipelining for High-Level Synthesis", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 32, No. 3, pp. 339-352, 2013

[11] K. Banerjee, C. Karfa, D. Sarkar, C. Mandal, "Verification of Code Motion Techniques Using Value Propagation", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 33, No. 8, pp. 1180-1193, 2014

[12] R. Sierra, C. Carreras, G, Caffarena, C. A. López Barrio, "A Formal Method for Optimal High-Level Casting of Heterogeneous Fixed-Point Adders and Subtractors", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 34, No. 1, pp. 52-62, 2015.

[13] S. Xydis, G. Palermo, V. Zaccaria, C. Silvano, "SPIRIT: Spectral-Aware Pareto Iterative Refinement Optimization for Supervised High-Level Synthesis", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 34, No. 1, pp. 155-159, 2015

[14] M. Dossis, "Intermediate Predicate Format for Design Automation Tools", Journal of Next Generation Information Technology, Vol. 1, No. 1, pp. 100-117, 2010

[15] M. Dossis, Patent number 1005308, 5/10/2006 by the Greek Industrial Property Organisation, 2006

[16] D. Amanatidis, M. Dossis, "High level synthesis of geometric active contours", 2nd Global Virtual Conference, Slovakia, April 7-11, 2014

[17] L. Wu, H. Huang, K. Su, S. Cai, X. Zhang, "An I/O Efficient Model Checking Algorithm for Large-Scale Systems", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 23, No. 5, pp. 905-915, 2015

[18] C. B. Harris, I. G. Harris, "Generating formal hardware verification properties from Natural Language documentation", 2015 IEEE International Conference on Semantic Computing (ICSC), Anaheim, CA, pp. 49-56, February 7-9, 2015

[19] R. A. Hernandez, M. Strum, W. J. Chau, "Transformations on the FSMD of the RTL code with combinational logic statements for equivalence checking of HLS", 16th Latin-American Test Symposium (LATS), Puerto Vallarta, Mexico, pp. 1-6, March 25-27, 2015

[20] V. Athavale, M. Sai, S. Hertz, S. Vasudevan, "Code coverage of assertions using RTL source code analysis", 51st ACM/EDAC/IEEE

Design Automation Conference (DAC), Moscone Center, San Francisco, CA, USA, pp. 1-6, June 1-5, 2014

[21] Yang Shuo, R. Wille, R. Drechsler, "Improving Coverage of Simulation-Based Verification by Dedicated Stimuli Generation", 17th Euromicro Conference on Digital System Design (DSD), Verona, Italy, pp. 599-606, August 27-29, 2014

AUTHORS PROFILE

Dr Michael Dossis holds an Advanced Engineering Diploma from NTUA, Greece, and a PhD in Electronic Engineering from University of Bradford, UK. He is the main author of a high number of international patents and quality publications in reputable international journals and conferences. Michael Dossis has a great experience as an ASIC project engineer and telecoms applications in companies such as LSI Logic Europe, ARM Ltd, Virata Ltd and Intracom Telecom S.A. He has experience in numerous programming and hardware description languages, and many design and CAD tools. His main interests include amongst others E-DA, High-level Synthesis and ESL, formal methods, AI applications and embedded VLSI systems.