Middle-East Journal of Scientific Research 22 (2): 233-240, 2014
ISSN 1990-9233

DOI: 10.5829/idosi.mejsr.2014.22.02.21851

Evaluation on High Level Synthesis for Parallel Computing in FPGA

[1]B. Saravana Kumaran, [2]M. Joseph

[1]Department of Information Technology, E.G.S. Pillay Engineering College, Nagapattinam, India
[2]Department of CSE, St. Joseph College of Engineering and Technology, Thanjavur, India

Abstract: The trends in technology are improving in almost all areas of application that varies from consumer goods, industrial application, electronic gadgets and communication. As the technological trends are improving the complexity in electronic design increases along with an increase in logical resources. These needs introduced a new technique of High Level Synthesis (HLS), in which the complexity in electronic design is reduced. These HLS tools are developed by many FPGA manufacturers based on the application needs and they compete among themselves for producing the tool that supports the faster time to market of the products. This paper analyzed the different HLS tools based on the feature they support. The tools are selected based on the availability of the license and this paper will give a glance on the tools for the beginner. The tools are evaluated on the basis of architectural support (*i.e*. FPGA, CPU, DSP and GPU), language support and Instruction Level Parallelism (ILP) and the results are tabulated.

Key words: High Level Synthesis (HLS) · Instruction Level Parallelism (ILP) · Architectural Support and Language Support

INTRODUCTION

The design of electronic circuit enters into the new era with the evolution of High Level Synthesis (HLS). The difficulties in design are reduced by progressing level of abstraction. The electronic circuit design flow can be viewed in three different ways, namely, geometry, behavior and structure as shown on Gajski and Kuhn's Y-chart [1]. In addition the five concentric circles in the chart represent Algorithm, Register Transfer Level (RTL), System level and Circuit and Logical representations. The algorithms elaborate the types of Programming languages, functions and loops. The operators and register are furnished by RTL concentric. In similar the system specification is represented by system level. Further the equation and logic representation are given by circuit and logic respectively. For the past twenty years, the hardware designer will manually develop the RTL from the scratch (*i.e.* from specification of behavioral system). From that instance the RTL synthesis followed by place and route will terminate the design flow. The flow should be verified at each and every stage, since; both tools and designs are prepared by the user. The errors and faults are minimized and reduced by taking action in the initial stages itself. As stated by Moore's law, the transistor size is doubled for every 1.5 years; by the way the challenge is increasing in the design flow. The design complexity is increased as the functionality of the design increase. This will lead to the increase in both design time and team (i.e. human resource). This confirms that, some techniques have to employ to enhance the productivity of the design. The design time is reduced by accepting the fact of the slightly.

Increased Area: The productivity of the design is improved by incorporating the High Level Synthesis (HLS), as it automatically converts from the algorithmic level to Register Transfer Level. The programming languages like 'C', 'C++' and Matlab are used to write the function based on which the HLS will generate the RTL design. The HLS reduce the designer's burden by taking as many tasks as possible from the user. The process of resource allocation is done after the code is analyzed. The time slot for each task is allotted in the scheduling process. At the end, the resource binding has two steps, such as, data element and operation of the source code are assigned to memory element and operator respectively. In addition to this, the peripheral interfaces

Corresponding Author: B. Saravana Kumaran, Department of Information Technology, E.G.S. Pillay Engineering College, Nagapattinam, India.

are monitored by the HLS, as it assigns the control and data signal between the peripheral and circuits. The HLS tools provide a number of positive progress towards the design flow, such as, it reduces the error in the code when compares to manual code. The time taken to write the code is reduced by HLS.

The recent research reveals that the time required for verification will be multiples of design time. In contrast to this, the HLS will reduce the verification time by generating the testbenches, since the test vector used in source code is used to verify the design. HLS is a promising candidate for embedded system based on FPGA. The design complexity has been reduced since the code for hardware architecture is maintained by HLS. The difficult hardware accelerator is designed using HLS with the least effort. The fastest time to market is achieved by the combination of FPGA and HLS. In this paper, the HLS tool is evaluated based on their suitability to the FPGA. Moreover the, tool is selected based on two parameters, namely, processing time and reduced delay. The advantage provide by the HLS made them modern research area and researchers have thrown a number of works based on HLS tool, which means the tools for HLS is developing rapidly. This paper evaluates the tools based on the availability of the license and their support to FPGA platform. This paper will give a basic startup to the one, who needs the faster prototyping in FPGA. The remaining paper is as follows, section 2 reveals the progress in HLS tools for FPGA, in section 3, the process of parallel computing in FPGA is discussed, the tools are evaluated based on architecture and language in section 4 and section5 concludes the paper.

Advancement of HLS Tool for FPGA: The HLS tools are practiced in real time environment, before the era of the FPGA. Hence, at the initial transformation the HLS tools are developed based on ASIC. A High Level Synthesis tool evolved in 1970 named as CMU-DA [2]. The tool flow and reduced complexity in CMU-DA attract the researchers and many tools are developed in the year 1980's and at the start of 1990. As a result of researcher's interest, the academician in California University developed Hyper/Hyper-LP [3] system. A new tool called HAL [4] was introduced by Bell-Northern research. University of Southern California developed the ADAM system [5]. Kiel University of Germany developed an HLS tool called as MIMOLA [6], apart from these tools Stanford University had lent a hand to develop a tool called as Hebe/Hercules [7, 8] HLS system. On the other hand, the interest from the semiconductor industries has contributed to many HLS tools. Like IBM develop a GM BSSC system [9] and Silicon Compiler [10]. All these tools are entertaining the same functions such as, generating the control, scheduling, binding and code transformation. From here the research is divided, as they are contributing to each individual problem, as HAL has force directed scheduling, in similar the conditional branch in the code is resolved by using a path-based scheduling in Silicon Compiler. Further the pipelined implementation is generated using the Sehwa tool in ADAM. The unbound delay in operation has been overcome by utilizing the relative scheduling technique in Hebe tool. The resource in the datapath is shared between the tasks by employing conflict Graph Colouring techniques. The design is specified using a custom language initially; Hercules system uses Hardware C to design the specification. This language is developed based on the feet of 'C' programming language. It will support the declarative and procedural semantics and mechanisms to support interface and design constraints. To enhance the efficiency of the DSP processor the Silage language is incorporated in the Cathedral/Cathedral-II. The language supports various and data types and transformation is made easy. In High Level Synthesis, the domain specific approach is initiated by combining the Cathedral-II and Silage programming language. Some real time chip is produced by utilizing these earlier tools, but among the designer community these tools are not widely adopted. Since the RTL synthesis was under the debate at that time and the RTL tools are not supported by the researchers. This scenario changes as the performance of the RTL tools, improved in 1990s. These tools are practically used as the semiconductor industries get attracted to the tools. Various tools are developed by Philips [11], Simens [12], IBM [13] and Motorola [14]. In addition to this the many HLS tools are provided by EDA vendors as Visual Architect from Cadence [15], Behavioral Compiler [16] from Synopsys and Mentor Graphics developed a tool called as Monte [17]. Since the system designers are not familiar with the behavioral HDL, these tools are not familiar among the design community as it uses behavioral HDL for specifying the input.

In relation to these tools, the debate among both research community and the design community is that using 'C' based language for HLS tools [18]. The demerits of 'C' language such as, timing, accuracy, interface, synchronization, hierarchy, parallelism and concurrency is overcome by developing the language based on 'C' language that would support hardware interface.

The hardware languages based on 'C' extension are Handel-C [19], SystemC [20], HardwareC [21] and SpecC [22]. Apart from these languages, some tools use various languages other than 'C' such as, Matlab [23] and BlueSpec [24]. The important factor that dominates the popularity of the HLS tools is that whether the tool can be targeted in Field Programmable Gate Array (FPGA). The modern day vendors developed many HLS tools that can support the FPGA like, Altera developed an FPGA targeted HLS tool, namely C2H [25], Mentor Graphics developed Handel-C compiler [19], ROCCC [26], CASH [27], Impulse C [28] and SPARK [29].

Parallel Computing in FPGA: The process of simultaneously computing the various logical, arithmetic and input/output operations is known to be parallel computing, it is based on the fact that the major problem is sub-divided into many sub-task and these sub-tasks are concurrently solved. The parallel computer varies accordingly, such as Grid Computing, Symmetric Multiprocessing (SMP), Cluster Computing, Graphical Processing Unit (GPU), Multicore Computing, Field Programmable Gate Array and parallel computing on ASIC's. The configuration of on chip RAM will connect the different logics and hardware resources in the FPGA. The configuration and reconfiguration of FPGA differs based on the time, if the SRAM is programmed at the start of the execution, then it is configuration and if SRAM is programmed during the execution then it is reconfiguration. These types of configuration support the users to implement the different algorithms and application on the same hardware. Also, this can even, change the logics and routing of an application at various times and leads to the Run-Time Reconfigured hardware. In tradition, the software written for the computer will compute the task in a serial manner. The problem is solved by constructing it as an algorithm and instruction set of the algorithm is executed serially. The central processing unit in the computer will execute this instruction as specified by the designer. In this approach, the processor will execute the instruction, one per second. The second instruction is executed after the completion of the first instruction. In contrast to serial computing, the problem is solved by multiple resources at a same time in parallel computing. The parallel computing is accomplished by mapping the algorithmic problem into independent part and executes each part independently with each other. These parts are also called as threads, process, task and threads based on the level of hierarchy. Coarse-grained parallelism exhibits, when the sub-task of the algorithm interact fewer time per second and Fine-grained parallelism exhibits, when the task interact many times per second [30-43].

The FPGA is scalable and it can run both fine-grained and coarse-grained architecture at the same time based on the application. Even the optimization of hardware based on the technology is possible as shown in [36]. The application with more complex algorithm cannot be implemented in single FPGA, in such case Multi-FPGA implementation can be used for executing the applications. However, this is not the case that differentiates FPGA from a traditional multiprocessor, the other features that differs FPGA are, reconfigurability of the FPGA is achieved by single instruction, interconnects can be reconfigured dynamically and the processing element is a single bit. In FPGA, the different level of parallelism is possible, such as, bit-level, data-level, pipelined-level, task-level and instruction-level parallelism also pipelined techniques can be implemented. Among which the instruction-level parallelism can contribute to the improvement in efficiency of FPGA in terms of delay (i.e. reduced delay).

***Recent Trends in Programming Tools for FPGA*:** The Hardware Description Language (HDLs) is being evolved from the past five decades and the Figure 1 shows an interesting view of the evolution of programming language.

The HDL language is designed to express the logic, algorithm and hardware object, moreover, it will execute the code in parallel manner in contrast the serial execution of conventional programming languages. The HDL language can be represented in two ways, such as Verilog and VHDL which had their own HDL language, having their own compiler, analyzer and syntaxes.

In distinguishing to this; the other HDL language is there, where their base is formed from programming languages like Java, Ruby, C, Occam, F, Matlab and C++. In the day-to-day development of the FPGA technology, the HDL language based second approach is recognized by the design community. This is because the designer can describe the specification in High Level of abstraction with very effectively with easy time to market.

Evaluation of the Tools

Based on FPGA: The need of hardware resource increases as the application needs are increasing. Hence, many FPGA vendors are competing among themselves to manufacture a high end board at low cost.

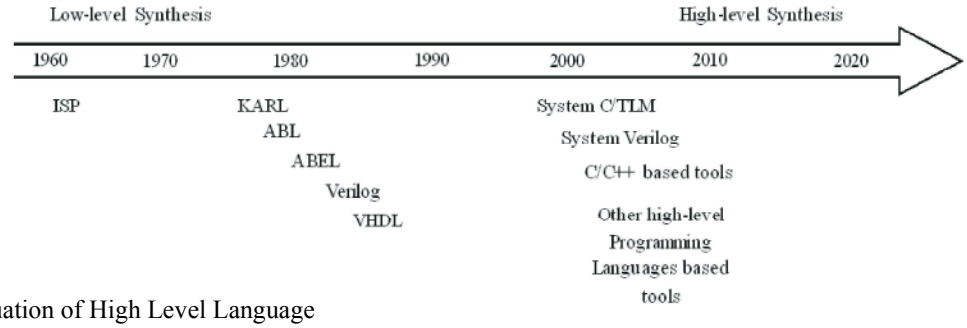

Fig. 1: Evaluation of High Level Language

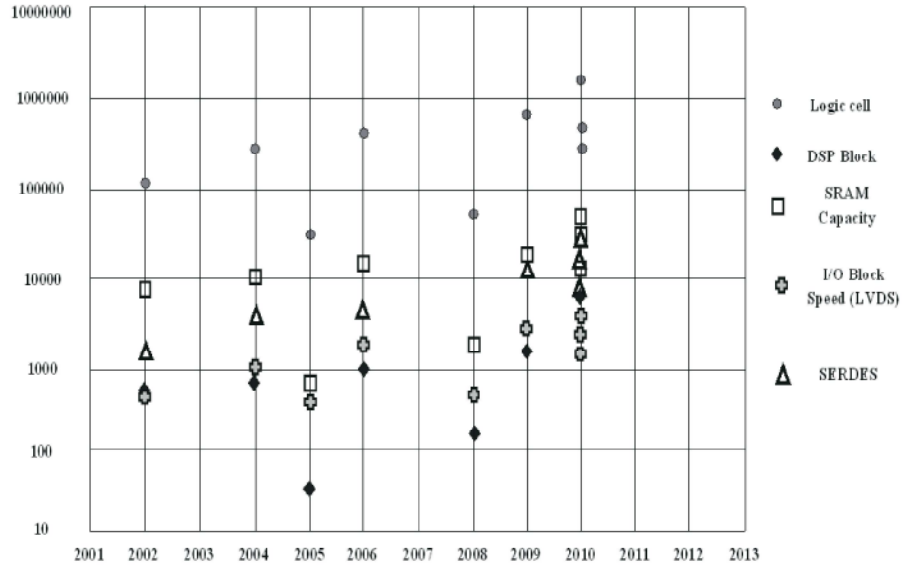

Fig. 2: Development in Xilinx FPGA

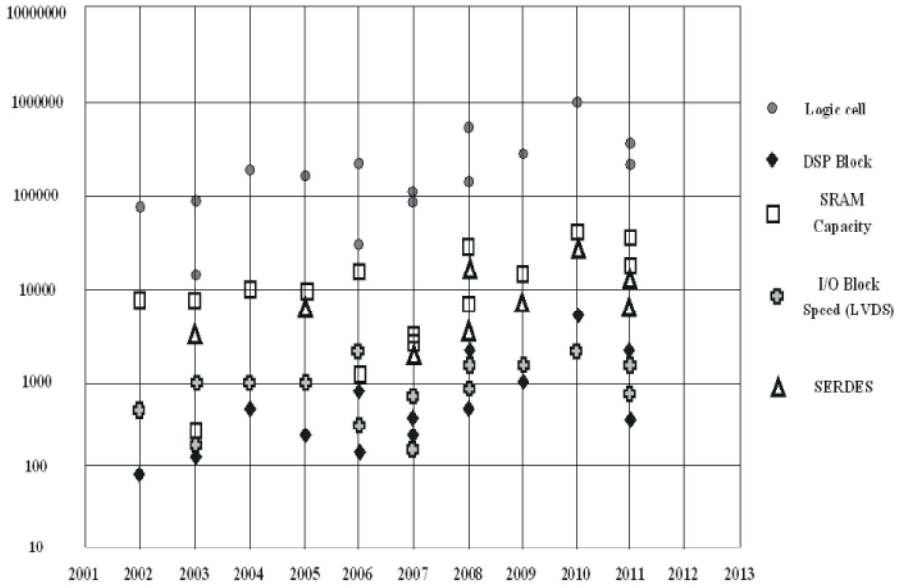

Fig. 3: Development in Altera FPGA

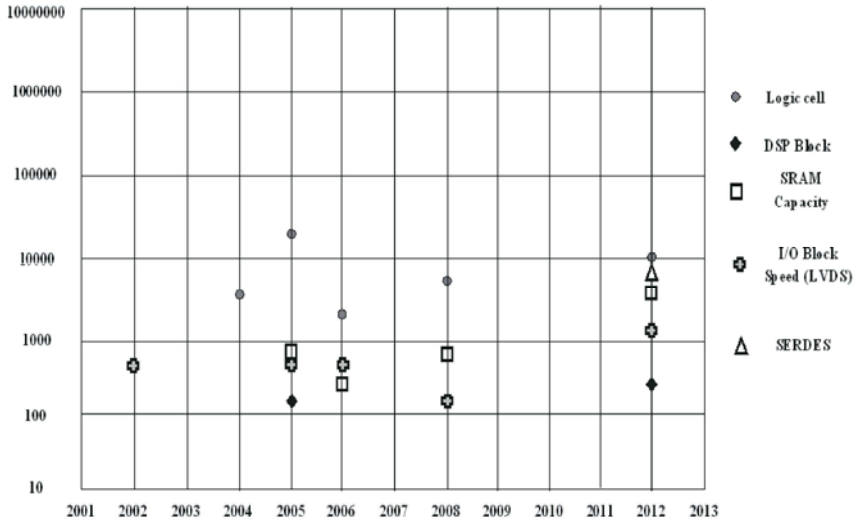

Fig. 4: Development in Actel FPGA

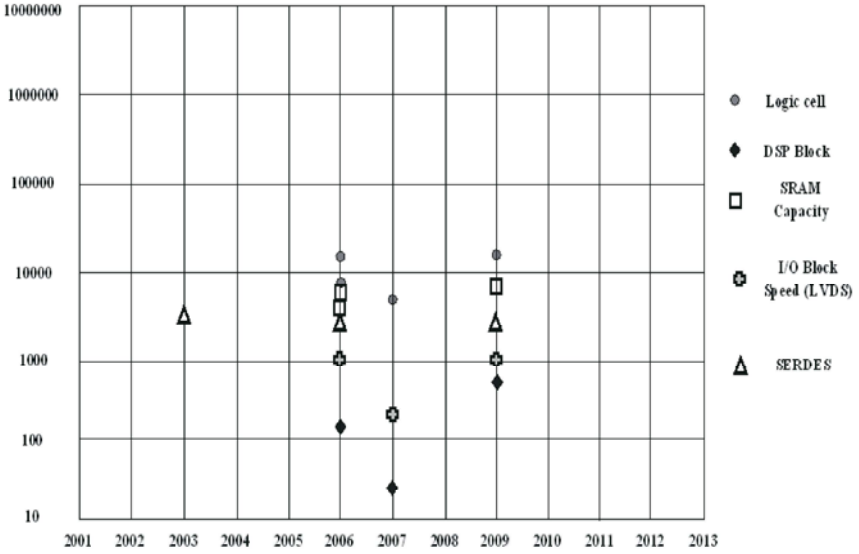

Fig. 5: Development in Lattice FPGA

The FPGA boards are classified based on the manufacturer also each manufacturer will have various boards based on their field of application they are going to be used. In similar, the boards can be classified based on their total number of logical resources, interfacing module, SRAM capacity, supportable DSP blocks, the speed of SERDES and I/O blocks. There are many semiconductor industries that manufacture the FPGA, among them some of the third party manufacturers are also there. But in recent years, there are four main FPGA competitor are there namely Xilinx, Altera, Lattice and Actel. Each manufacturer introduces their own boards along with several features at various costs. The cost varies accordingly with the configuration of the board and application they are made for. The Figure 2 shows the Xilinx board developed over the year in terms of logical resource and speed, in similar the same result for Altera board is shown in Figure 3. The basic boards used in the research, institution and in industries are listed and the same count for Actel and Lattice is shown in Figure 4 and Figure 5 respectively [40].

***Based on Language Support*:** HLS tools can also be evaluated on the various features of their language. The supporting language should feature the designer with High Level Programming language (HLPL) to specify

Table 1: Open Tools Supporting Various Architecture

References	Tool Name	Proprietary	Support of FPGA/CPU/GPU	Feature Support	Instruction Level Parallelism
[37]	LegUP	Open	DSP	HLPL, LU, AP, LP	Yes
[38]	CUDA	Open	CPU	HLPL	Yes
[39]	Open HMPP	Open	CPU, GPU	HLPL	No
[40]	MPI	Open	CPU	HLPL	Yes
[41]	Open ACC	Open	CPU, GPU	HLPL	Yes
[42]	Open MP	Open	CPU	HLPL	No
[43]	POSIX threads	Open	CPU	HLPL	Yes

Table 2: Proprietary Tools Supporting FPGA Architecture

References	Tool Name	Proprietary	Support of FPGA/CPU/GPU	Feature Support	Instruction Level Parallelism
[30]	Vivado	Xilinx	FPGA	GM, HLPL, SCPE, LU, AP, LP	Yes
[31]	Open CL	Altera	FPGA	HLPL, SCPE,LU, AP, LP	Yes
[23]	Matlab Coder	Math Works	FPGA	GM, HLPL, SCPE, LU, AP, LP	Yes
[19]	Handel-C	Mentor Graphics	FPGA	HLPL, SCPE,LU, AP, LP	Yes
[28]	ImpulseC	Impulse AT	FPGA	HLPL, SCPE,LU, AP, LP	Yes
[32]	Icarus Verilog	Open	FPGA	HLPL, SCPE, LU, LM, LP	Yes
[33]	Cynthesizer	Forte DS	FPGA	HLPL, SCPE, LU, AP, LP	Yes
[34]	C-to-Silicon Compiler	Cadence	FPGA	HLPL, SCPE, LU, AP, LP	Yes
[35]	Synphony C Compiler	Synopsys	FPGA	HLPL,SCPE,LU, AP, LP	Yes

the algorithm. The Abstract Syntax Tree (AST) formation of the initial source code should be formulated in parsing stage; the scheduling stage in the HLS tools should support Loop Unrolling (LU), Loop Merging (LM), Hierarchical Synthesis (HS), Loop Pipelining (LP), Automatic Pipelining (AP), the final binding stage will allocate each operation of a hardware unit. Apart from these features, the architecture support of the tools based on Field Programmable Gate Array (FPGA) and language are considered. Another important characteristic is the tool should support reduced delay and high speed of the application and Instruction Level Parallelism (ILP).

The Table 1 shows the tools that support different architecture such as Central Processing Unit (CPU), Digital Signal Processing (DSP) and Graphics Processing Unit (GPU). The result in the Table 1 is analyzed with the open tools and there is no support for High Level Synthesis. The tool that supports the FPGA architecture is shown in Table 2 with their proprietary. The last column in the table shows the language supporting the Instruction Level Parallelism (ILP), which is one of the major areas of research in High Level Synthesis (HLS). Since these tools are developed from a major competitor, the tools support various algorithms and techniques that would increase the performances of the application developed in High Level Synthesis (HLS).

CONCLUSION

This paper presents a number of various High Level Synthesis (HLS) tools with various feature and parameters support by the tools. Also, analyzed the various High Level Language (HLL) both based on ‘C’ programming and individual Hardware Description Language (HDL).

The tools presented in the evaluation section are analyzed under different categories such as the architectural support (i.e. CPU, DSP, FPGA, GPU) and language support (i.e. Instruction Level Parallelism). The ILP is an important feature that would reduce the delay and increase the speed of operation. However, the tools analyzed in this paper are based on the availability of the license and this paper can give a modest introduction to the tools that supports ILP and in the future the idea of Instruction Level Parallelism (ILP) is progressing and taken to the next stage by applying it in some applications.

REFERENCES

1. Gajski, D.D. and R.H. Kuhn, 1983. New VLSI tools, Computer, 16: 11-14.
2. Parker, A., D. Thomas, D. Siewiorek, M. Barbacci, L. Hafer, G. Leive and J. Kim. The CMU design automation system: an example of automated data path design, in Proc. DAC, 79: 73-80.
3. Rabaey, J., C. Chu, P. Hoang and M. Potkonjak, 1991. Fast prototyping of datapath-intensive architectures, IEEE Design and Test, 8(2): 40-51.
4. Paulin, P.G., J.P. Knight and E.F. Girczyc. HAL: A multi-paradigm approach to automatic data path synthesis. in Proc. DAC, 86: 263-270.
5. Granacki, J., D. Knapp and A. Parker, The ADAM advanced design automation system: overview, planner and natural language interface, in Proc. DAC, 85: 727-730.
6. Marwedel, P. The MIMOLA design system: tools for the design of digital processors, in Proc. DAC, 84: 587-593.
7. Micheli, G. De, D. Ku, F. Mailhot and T. Truong, 1990. The Olympus synthesis system, IEEE Design and Test of Computers, 7(5): 37-53.
8. Micheli, G. De and D. Ku, HERCULES-A system for high-level synthesis, in Proc. DAC, 88: 483-488.
9. Yassa, F.F., J.R. Jasica, R.I. Hartley and S.E. Noujaim, 1987. A silicon compiler for digital signal processing: methodology, implementation and applications, in Proc. IEEE, 7(9): 1272-1282.
10. Composano, R. Design process model in the Yorktown Silicon Compiler. in Proc. DAC, 88: 489-494.
11. Lippens, P.E.R., J.L. van Meerbergen, A van der Werf, W.F.J. Verhaegh, B.T. McSweeney, J.O. Huisken and O.P. McArdle. PHIDEO: a silicon compiler for high speed algorithms, in Proc. EDAC, 91: 436-441.
12. Biesenack, J., M. Koster, A. Langmaier, S. Ledeux, S. Marz, M. Payer, M. Pilsl, S. Rumler, H. Soukup, N. Wehn and P. Duzy. The Siemens high-level synthesis system CALLAS, IEEE Trans. VLSI Systems, 1(3): 244-253.
13. Bergamaschi, R.A, R.A. O' Connor, L. Stok, M.Z. Moricz, S. Prakash, A. Kuehlmann and D.S. Rao. High-level synthesis in an industrial environment, IBM Journal of Research and Development, 39(1-2): 131-148.
14. Küçükçakar, K., C.T. Chen, J. Gong, W. Philipsen and T.E. Tkacik. Matisse: an architectural design tool for commodity ICs. IEEE Design and Test of Computers, 15(2): 22-33.
15. Hemani, A., B. Karlsson, M. Fredriksson, K. Nordqvist and B. Fjellborg. Application of high-level synthesis in an industrial project. in Proc. VLSI Design, 94: 5-10.
16. Knapp, D.W., 1996. Behavioral synthesis: digital system design using the Synopsys Behavioral Compiler. Prentice-Hall.
17. Elliott, J.P., 1999. Understanding behavioral synthesis: a practical guide to high-level design. Springer.
18. Edwards, S.A., 2006. The challenges of synthesizing hardware from C-like Languages, IEEE Design and Test of Computers, 23(5): 375-386.
19. Agility Design Solutions, 2007. Handel-C language reference manual.
20. IEEE and OCSI, 2005. IEEE 1666TM-2005 Standard for SystemC. http://www.systemc.org.
21. Ku, D. and G. De Micheli, 1990. Hardware C-a language for hardware design (version 2.0), Technical Report. UMI Order Jumber: CSL-TR-90-419. Stanford University.
22. Gajski, D., J. Zhu, R. Dömer, A. Gerstlauer and S. Zhao, 2000. Spec: specification language and methodology. Kluwer Academic Publishers.
23. Haldar, M., A. Nayak, A. Choudhary and P. Banerjee. A system for synthesizing optimized FPGA hardware from MATLAB. in Proc. ICCAD, 01: 314-319.
24. BlueSpec, Inc. http://www.bluespec.com.
25. Altera Corporation, 2009. Jios II C2H compiler user guide, version, 9: 1.
26. Villarreal, J., A. Park, W. Najjar and R. Halstead. Designing modular hardware accelerators in C with ROCCC 2.0. in Proc. FCCM, 10: 127-134.
27. Budiu, M., G. Venkataramani, T. Chelcea and S. Goldstein. Spatial computation, in Proc. ASPLOS, 04: 14-26.

28. Pellerin, D. and S. Thibault, 2005. Practical FPGA programming in C. Prentice Hall Professional Technical Reference.
29. Gupta, S., R. Gupta, N. Dutt and A. Nicolau, 2004. SPARK: a parallelizing approach to the high-level synthesis of digital circuits, Springer.
30. The Xilinx Vivado homepage. http://www.xilinx.com/ products/design-tools/vivado/.
31. The opencl homepage. https://developer.nvidia.com/ opencl.
32. http://iverilog.icarus.com/
33. The cynthesizer 5 homepage. http://www.forteds.com/products/ cynthesizer.asp.
34. The c-to-silicon homepage." http://www.forteds.com/ products/cynthesizer.asp.
35. http://www.synopsys.com/Systems/BlockDesign/H LS/Pages/SynphonyC-Compiler.aspx.
36. Joseph, M., Bhat Narasimha. B. Sekaran and K. Chandra, 2007. Technology driven High-Level Synthesis. International Conference on ADCOM, pp: 485-490.
37. The legup homepage." http://legup.eecg.utoronto.ca /.
38. The cuda homepage." http://www.nvidia.com/ object/cuda home new.html.
39. The openhmpp homepage." http://www.caps-entreprise.com/openhmpp-directives/.
40. The open mpi homepage." http://www.open-mpi.org/.
41. The openacc homepage." http://www.openacc.org/.
42. The openmp homepage." http://openmp.org/.
43. The posix threads tutorial."https:// computing.llnl.gov/tutorials/pthreads/.