A Survey of High Level Synthesis Languages, Tools, and Compilers for Reconfigurable High Performance Computing

Luka Daoud[1], Dawid Zydek[1], and Henry Selvaraj[2]

[1] Department of Electrical Engineering, Idaho State University, Pocatello, ID, USA
{daouluka,zydedawi}@isu.edu
[2] Department of Electrical and Computer Engineering, University of Nevada, Las Vegas, NV, USA
henry.selvaraj@unlv.edu

Abstract. High Level Languages (HLLs) make programming easier and more efficient; therefore, powerful applications can be written, modified, and debugged easily. Nowadays, applications can be divided into parallel tasks and run on different processing elements, such as CPUs, GPUs, or FPGAs; for achieving higher performance. However, in the case of FPGAs, generating hardware modules automatically from high level representation is one of the major research activities in the last few years. Current research focuses on designing programming platforms that allow parallel applications to be run on different platforms, including FPGA. In this paper, a survey of HLLs, tools, and compilers used for translating high level representation to hardware description language is presented. Technical analysis of such tools and compilers is discussed as well.

Keywords: High Level Synthesis, Compilers, FPGA, C-to-VHDL.

1 Introduction

Today, the trend in High Performance Computing (HPC) is to run applications in parallel on different processing elements. Applications can be divided into multiple tasks and executed simultaneously. Performance of such parallel processing systems has increased significantly over the past few years and that trend continues till date. The improvement is possible by implementing the multi-core versions of conventional CPUs (Central Processing Units), and by hybrid computing platforms with accelerators, such as FPGAs (Field Programmable Gate Arrays) or GPUs (Graphics Processing Units) [1,2].

In general, applications and simulations [3] are not well suited for executing exclusively on accelerators. Some portions of applications have extensive parallelism, suitable for FPGAs or GPUs; others are inherently serial or have extensive control flow that make them better suited for a CPU or again for an FPGA. Such device-oriented application partitioning increases the overall system performance.

J. Świątek et al. (eds.), *Advances in Systems Science*,
Advances in Intelligent Systems and Computing 240,
DOI: 10.1007/978-3-319-01857-7_47,

Hybrid HPC is a promising approach to increase the performance of supercomputers. It combines computing platforms, such as CPUs, GPUs, and FPGAs; in order to attain higher performance. FPGAs offer energy efficiency and higher throughput for portions of applications characterized by simple data objects and extensive parallelism. GPUs outperform FPGAs for streaming and floating-point applications; and for applications requiring high memory bandwidth. CPUs are optimized for serial processing. Combining these computing platforms ensures HPC and decreases energy consumption. The main challenge in recent HPC software is to design a solution that allows using CPU (as a main processing unit), GPUs, FPGAs, and other accelerators. Moreover, the software should be able to decide which portion of the application is suitable for which computing platform, considering higher performance and energy efficiency (Fig. 1).

In order to implement application code on an FPGA, the code should be written in Hardware Description Language (HDL), like e.g. in [4] or [5]. Due to rapid increase of complexity in the systems, researchers and engineers moved from Register Transfer Level (RTL) design to high level design, seeking better productivity in less time and with lower cost. Therefore, this paper provides a survey on current and recent High Level Synthesis (HLS) tools, languages, and compilers for reconfigurable systems. These HLS compilers are categorized in this paper based on the input language. This paper is a first step in developing a new tool that translates High Level Language (HLL) to a code recognizable by variety of computing hardware, such as CPUs, GPUs, FPGAs, DSPs (Digital Signal Processors), or others. The paper is organized as follows: The next section compares FPGAs to CPUs and GPUs. A survey on HLS, languages, and compilers is presented in Section 3; and finally Section 4 concludes the paper.

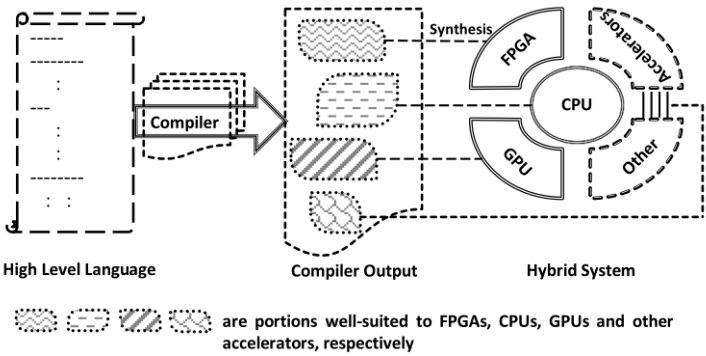

Fig. 1. A compiler compiling an application-code for CPUs, GPUs, FPGAs, and other accelerators

2 FPGAs vs. CPUs and GPUs

FPGAs and/or GPUs are used in hybrid computing systems to accelerate the computing processes [1]. Hence, HPC can be obtained. In HPC systems, the FPGA acts as a configurable co-processor to a CPU, where FPGA executes intensive computational parts of the code. Similarly, GPU processes large blocks of data in parallel to increase performance. Although FPGAs run at frequencies expressed in MHz while CPUs run at few GHz, very often FPGAs outperform CPUs. The reasons behind that are:

- For each specific task a dedicated circuit is implemented in the FPGA,
- Designers exploit the parallelism and pipeline of the circuit implemented on the FPGA,
- FPGAs offer huge memory bandwidth through configurable logic.

Besides higher performance offered by FPGAs, they provide lower power consumption in comparison to CPUs. In HPC systems, GPUs are intensively used for numerous scientific applications to achieve higher performance by off-loading the most intensive computing portions of the application to the GPUs. GPUs often outperform FPGAs for streaming applications. They usually have a higher floating-point performance and memory bandwidth than FPGAs. However, according to [6], FPGAs present better computing capability for applications characterized by:

- Relatively simple data objects,
- Relatively simple arithmetic operations,
- Smooth implementation using pipelined processing structures,
- Extensive data-parallelism,
- Regular and simple control structures.

3 HLS Languages, Tools, and Compilers

In this section, we analyze HLS tools and programming languages for FPGAs. The tools enhance the portability and scalability of applications. Moreover, they optimize performance and design efforts as well. Most of these tools are based on C/C++ programming language, because the language is well known for both software and hardware engineers. Fig. 2. shows the flow of generating RTL from HLL.

3.1 HLS from C/C++ Programming Language

Hardware-C [7] is one of the first HLS languages that uses the C programming language. It supports parallel processes that communicate together through either port passing or message passing techniques. However, it cannot represent arbitrary serial-parallel structures and it has different syntax from the original C for several constructs. Handel-C [8] is one of the HLLs based on C language,

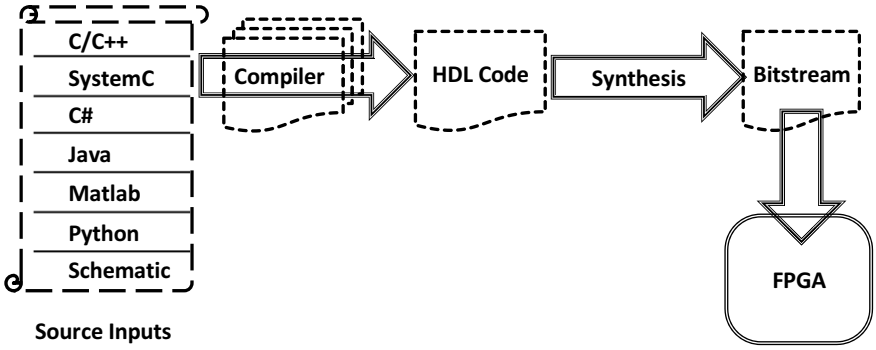

Fig. 2. The flow of generating RTL from HLL

where the commands are written one by one and they are executed sequentially. Handel-C targets low level hardware where the commands can be executed in FPGA. In order to get benefit of parallel execution, some keywords are used in the code. Therefore, performance benefits can be attained by using parallelism. The overall program structure of Handel-C is little different from conventional C. The program structure consists of one or more main functions, each associated with a clock. Thus parts of the program can be run at different speeds. When a code is written in Handel-C, the programmer should be aware of the hardware implementations in order to get the benefit of parallelism. Similar to Handel-C, Hyden-C [9] is a framework of optional annotations to enable designers to describe design-constraints and to direct source-level transformations such as scheduling and resource allocation. The main difference between Handel-C and Hyden-C is that Hyden-C, like VHDL, is component-based. Therefore, designers can describe their designs as a set of distinct components that are developed independently and then connect them together.

Many HLS tools are designed for application domains. For example, Trident [10] is a compiler that accepts C code extracting the parallelism and the possibility of pipeline implementations from the code; and generates the corresponding circuits in reconfigurable logic. Trident compiler is mainly designed for floating-point applications. GAUT [11] is an academic HLS tool dedicated to DSP applications. The GAUT tool converts a C function into a pipelined architecture consisting of a processing unit, memory unit, communication, and multiplexing unit. Also, Streams-C [12] and Impulse-C (derived from Streams-C) [13] are compilers that support stream-oriented computation on FPGA-based parallel processors, where data parallelism can be effectively mapped onto the FPGA.

In addition, many of the free and open source online compilers can be used to convert the C code (HLL) to HDL. C to Verilog [14] is an online compiler that translates the C function into a hardware-module interface. Although this compiler uses most of the C language features, there are some limitations in the C code that are not acceptable by the tool, e.g. recursive functions, structures, pointers to functions, and library function calls (printf, malloc, etc). These

structures (limitations) cannot be represented in hardware. FpgaC [15] is a compiler for a subset of the C programming language. It produces digital circuits that execute the compiled programs.

Since hybrid systems provide higher performance, some compilers target hardware and software systems by translating specific parts of a larger C program into hardware; and the rest of the program is executed on a traditional CPU. NAPA C [16] is a hardware-software and co-synthesis compiler that generates a C program targeting hybrid RISC CPUs and FPGAs. Similarly, Nimble [17] is a framework that automatically compiles system-level applications specified in C to the code executable on the combined CPU and FPGA architectures. Also, CHiMPS [18] is a C-based compiler for hybrid CPU-FPGA computing platform. Similar to CHiMPS, CASH [19] is a compiler that targets the hybrid System on Chips (SoCs). LegUp [20] is an open source HLS tool that automatically compiles a C program to target a hybrid FPGA-based software/hardware system. The program can be divided into program segments, some program segments are executed on an FPGA-based MIPS CPU and other program segments are automatically synthesized into FPGA circuits. These circuits communicate and work together with the CPU. ROCCC [21] is an open source compiler that accepts a strict subset of C and generates VHDL. In order to be used efficiently, the entire software program is not translated into hardware – ROCCC focuses on the critical regions of software, e.g. regions containing loops performing extensive computation on large amounts of data. The Nios II C-to-Hardware Acceleration (C2H) compiler [22] is a tool that allows the designer to create custom hardware accelerators directly from C code. Altera's C2H allows partitioning C functions into hardware sets that can be executed on a Nios II CPU. By using the C2H compiler, an algorithm in C targeting a Nios II CPU can be quickly converted to a hardware accelerator implemented on an Altera's FPGA.

There are also C++ language compilers that can be used to generate HDL. A Stream Compiler (ASC) [23] is a C++ library allowing the designers to optimize the hardware implementation on the algorithm level, architecture level, arithmetic level, and gate level; all within the same C++ program. ASC code is compiled to produce hardware netlist circuit. Catapult C [24] is a subset of C++ with no extensions. It takes ANSI C/C++ or SystemC [25] as input and generates RTL code targeting FPGAs or ASICs. The code that can be compiled from Catapult C may be very general and may result in many different hardware implementations. AutoPilot [26] is one of the most recent HLS tools. It automatically generates efficient RTL code from high level representations. Autopilot accepts three kinds of standard C-based design entries: C, C++, and SystemC. AutoPilot is an advanced compiler capable of carrying out efficient power optimization using clock gating and power gating. It also supports pipeline to improve the system performance. Hence, it can target a wide range of applications.

In addition, DEFACTO [27] and Carte [28] are compilers that accept C and Fortran as input languages. DIME-C [29] is a C based compiler that translates a DIME-C code into VHDL. However, like C to Verilog [14], not all elements in C

languages are supported in DIME-C, e.g. pointers, structures, switch statements, etc. Other C-based compilers include Bash-C [30], Mitrion-C [31], SpecC [32], SPC [33], or SPARK [34]. Additional commercial tools and early compilers can be found in [35].

3.2 HLS from Non-C/C++ Programming Language

MATlab Compiler for Heterogeneous computing systems (MATCH) [36] allows users to develop efficient codes for distributed, heterogeneous, and reconfigurable computing systems. MATCH takes MATlab programs and automatically maps them onto a hybrid computing environment consisting of embedded CPUs, DSPs, or FPGAs.

MyHDL [37] is an open source Python package that allows the designer using Python to generate HDL. The Python code is converted to Verilog and VHDL.

JHDL (Just-Another Hardware Description Language)[38] is a design tool for reconfigurable systems that focuses mainly on designing circuits through an object oriented approach. The main use of JHDL is to create digital circuits for implementation using FPGAs. JHDL can be used with any standard Java 1.1 distribution without language extensions. Also, based on Java source input, Sea Cucumber (SC) [39] is a synthesizing compiler for FPGAs that accepts Java class files as input and then generates circuits. Users write circuit descriptions exposing coarse-level parallelism as concurrent threads. SC then analyzes the body of each thread and uses compiler and circuit optimization techniques to extract fine-grained parallelism. Afterwards, SC compiler is executed to generate an Electronic Design Interchange Format (EDIF) netlist; and the Xilinx place and route software is called to create a bitstream from the synthesized EDIF netlist.

Kiwi [40] is a compiler based on C#. The Kiwi compiler accepts common intermediate language output from either the .NET or Mono C# compilers and generates Verilog RTL.

Pebble [41] is a language for parameterized and reconfigurable hardware design. Pebble has a simple block-structured syntax. Designers can easily define the number of pipeline stages using the parameters in a Pebble program. The main objective of Pebble is to support the development of designs involving run-time reconfiguration.

There are also other HLS compilers that use different languages. For example, Esterel [42] is a synchronous programming language designed to program reactive systems (systems that react continuously to their environment). Another example is BlueSpec compiler [43] that works based on Bluespec System Verilog – a language used in the design of electronic systems.

3.3 Schematics-Based HLS

Schematics such as LabVIEW and MATlab are also used to program FPGAs. LabVIEW is one of the schematic tools that targets FPGAs. The National Instruments (NI) LabVIEW FPGA module extends the LabVIEW graphical

development platform to target FPGAs on NI reconfigurable I/O hardware. Since LabVIEW represents parallelism and data flow, it is suitable for FPGA programming [44]. In addition, designers also can use MATlab to design and simulate their algorithms by Simulink and Stateflow, then MATlab generates VHDL or Verilog code for FPGAs using HDL Coder [45]. Another tool is Altium Designer [46]. It is an electronic design automation software package for printed circuit board, FPGA, and embedded software design.

3.4 HLS Based on Programming Models for GPUs

Parallel computing platforms and programming models for GPUs are subject of very intense research. CUDA (Compute Unified Device Architecture) and OpenCL (Open Computing Language) are parallel programming models that address the higher interest in GPUs; moreover, they have recently expanded their capabilities beyond GPUs.

CUDA is a parallel computing platform and programming model created by NVIDIA and implemented on their GPUs. FCUDA [47] is a framework to convert CUDA code to RTL suitable for FPGAs. The transformation process from CUDA to RTL is done in two phases. First, FCUDA transforms the single-program-multiple-data CUDA code into C code for AutoPilot [25] with annotated coarse-grained parallelism. Then, the AutoPilot maps the marked parallelism onto parallel cores and generates the corresponding RTL description. Afterwards, synthesis and programming of FPGA is done. The main goal of the FCUDA is to convert thread blocks into C functions. FCUDA combines the CUDA programming model with a HLS tool (AutoPilot) to efficiently implement CUDA code on FPGA.

Another parallel programming framework for writing programs that are executed across heterogeneous platforms is OpenCL [48]. SOpenCL [49] is an OpenCL-based FPGA synthesis tool. It generates hardware circuits and SoC systems from OpenCL programs. The output of SOpenCL is a pure C function which is converted to a hardware circuit in a form of synthesizable HDL.

On the other hand, Altera enables the designers to run OpenCL code on Altera's FPGAs [50]. Compiling an OpenCL code to FPGA by Altera's solution is the process of converting an OpenCL C code into FPGA bitstream that allows programming the FPGA. The compilation process has two phases: the OpenCL code is compiled into intermediate hardware format code, and then compiled into an FPGA bitstream. Therefore, each OpenCL code is converted into custom hardware representing the data flow circuit. Mapping multithreaded functions to FPGA can be done simply by replicating hardware (inefficient – waste of resources) or by using pipeline parallelism (more efficient mapping).

4 Final Remarks

The trend in HPC is to increase the number of processing elements using heterogeneous computing platforms, in order to provide higher performance and

increase energy efficiency. Since the complexity in applications and systems has been increasing, designers move to high level representation to improve the product quality in less time and with lower cost.

This paper is a survey of HLS tools and compilers that accept HLL code and generate HDL or bit-stream files for FPGAs. These HLS tools and compilers have been presented according to the input source code. Most current and recent compilers have been presented. Also, compilers that convert a code for GPUs, written in CUDA or OpenCL, to RTL form have been demonstrated in this paper. This work has been presented as the first step to design a compiler capable of compiling and analyzing code for heterogeneous systems combining CPUs, GPUs, and FPGAs. As a future work, we plan to design such a compiler that will use intelligent techniques to select the best computing platform for all portions of the code, in order to increase performance. Our future work will also focus on improving efficiency of FPGAs for floating-point calculations.

References

1. Liu, B., Zydek, D., Selvaraj, H., Gewali, L.: Accelerating High Performance Computing Applications Using CPUs, GPUs, Hybrid CPU/GPU, and FPGAs. In: Proceedings of the 13th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2012), pp. 337–342 (2012), doi:10.1109/PDCAT.2012.34
2. Zydek, D., Selvaraj, H., Borowik, G., Luba, T.: Energy Characteristic of Processor Allocator and Network-on-Chip. International Journal of Applied Mathematics and Computer Science 21(2), 385–399 (2011), doi:10.2478/v10006-011-0029-7
3. Chmaj, G., Zydek, D.: Software development approach for discrete simulators. In: Proceedings of the 21st International Conference on Systems Engineering (ICSEng 2011), pp. 273–278. IEEE Computer Society Press (2011), doi:10.1109/ICSEng.2011.56
4. Zydek, D., Selvaraj, H., Gewali, L.: Synthesis of Processor Allocator for Torus-based Chip Multiprocessors. In: Proceedings of the 7th International Conference on Information Technology: New Generations (ITNG 2010), pp. 13–18. IEEE Computer Society Press (2010), doi:10.1109/ITNG.2010.145
5. Zydek, D., Selvaraj, H.: Hardware Implementation of Processor Allocation Schemes for Mesh-based Chip Multiprocessors. Microprocessors and Microsystems 34(1), 39–48 (2010), doi:10.1016/j.micpro.2009.11.003
6. Chase, J., Nelson, B., Bodily, J., Wei, Z., Lee, D.: Real-time Optical Flow Calculations on FPGA and GPU Architectures: A Comparison Study. In: Proceedings of the 16th International Symposium on Field-Programmable Custom Computing Machines, FCCM 2008, pp. 173–182 (2008)
7. Ku, D.C., De Micheli, G.: Hardware C - A Language for Hardware Design. Technical report, DTIC Document (1988)
8. Aubury, M., et al.: Handel-C Language Reference Guide. Computing Laboratory. Oxford University, UK (1996)
9. Coutinho, J., Luk, W.: Source-directed Transformations for Hardware Compilation. In: Proceedings of the International Conference on Field-Programmable Technology (FPT), pp. 278–285. IEEE (2003)

10. Tripp, J., et al.: Trident: An FPGA Compiler Framework for Floating-Point Algorithms. In: Proceedings of the International Conference on Field Programmable Logic and Applications, pp. 317–322 (2005)
11. GAUT- High-Level Synthesis Tool From C to RTL (May 2013), http://hls-labsticc.univ-ubs.fr
12. Gokhale, M., Stone, J., Arnold, J., Kalinowski, M.: Stream-Oriented FPGA Computing in the Streams-C High Level Language. In: 2000 IEEE Proceedings of the Symposium on Field-Programmable Custom Computing Machines, pp. 49–56 (2000)
13. http://www.impulseaccelerated.com/ReleaseFiles/Help/ImpulseCUserGuide.pdf: (May 2013)
14. C to Verilog (May 2013), http://www.c-to-verilog.com
15. FpgaC Compiler (May 2013), http://www.utb.edu/vpaa/csmt/cis/Pages/FPGAc.aspx
16. Gokhale, M., Stone, J.: NAPA C: Compiling for a Hybrid RISC/FPGA Architecture. In: Proceedings of the IEEE Symposim on FPGAs for Custom Computing Machines, pp. 126–135 (1998)
17. Li, Y., et al.: Hardware-Software Co-Design of Embedded Reconfigurable Architectures. In: Proceedings of the 37th Annual Design Automation Conference, pp. 507–512 (2000)
18. Putnam, A., et al.: CHIMPS: A C-Level Compilation Flow for Hybrid CPU-FPGA Architectures. In: 2008 FPL, Proceedings of the International Conference on Field Programmable Logic and Applications, pp. 173–178 (2008)
19. Budiu, M., Goldstein, S.C.: Compiling Application-Specific Hardware. In: Glesner, M., Zipf, P., Renovell, M. (eds.) FPL 2002. LNCS, vol. 2438, pp. 853–863. Springer, Heidelberg (2002)
20. Canis, A., et al.: LegUp: High-Level Synthesis for FPGA-based Processor/Accelerator Systems. In: Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays, pp. 33–36 (2011)
21. Villarreal, J., Park, A., Najjar, W., Halstead, R.: Designing Modular Hardware Accelerators in C with ROCCC 2.0. In: Proceedings of the 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 127–134 (2010)
22. http://www.altera.com/literature/ug/ug_nios2_c2h_compiler.pdf (May 2013)
23. Mencer, O.: ASC: A Stream Compiler for Computing with FPGAs. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 25(9), 1603–1617 (2006)
24. Mentor Graphics, Catapult C Synthesis (May 2013), http://www.mentor.com
25. Initiative, Open SystemC: SystemC 2.0. 1 Language Reference Manual. Revision 1(1177), 95118–3799 (2003)
26. Zhang, Z., Fan, Y., Jiang, W., Han, G., Yang, C.: AutoPilot: A platform-based ESL Synthesis System. In: High-Level Synthesis, pp. 99–112. Springer (2008)
27. Bondalapati, K., et al.: DEFACTO: A Design Environment for Adaptive Computing Technology. Springer (1999)
28. Poznanovic, D.S.: Application Development on the SRC Computers, Inc. Systems. In: Proceedings of the 19th IEEE International Symposium on Parallel and Distributed Processing, pp. 1–10 (2005)
29. Park, S., Shires, D., Henz, B.: Reconfigurable Computing: Experiences and Methodologies. Technical report, DTIC Document (2008)

30. Yamada, A., et al.: Hardware Synthesis with the Bach System. In: Proceedings of the 1999 IEEE International Symposium on Circuits and Systems, ISCAS 1999, vol. 6, pp. 366–369 (1999)
31. Mitrionics AB: Mitrion Users Guide. Technical report, Mitrionics (2008)
32. Domer, R.: The SpecC System-Level Design Language and Methodology, Part 1, Parts 1 & 2. In: Embedded Systems Conference (2001)
33. Weinhardt, M., Luk, W.: Pipeline Vectorization. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 20(2), 234–248 (2001)
34. Gupta, S., Dutt, N., Gupta, R., Nicolau, A.: SPARK: A High-Level Synthesis Framework for Applying Parallelizing Compiler Transformations. In: Proceedings of the 16th International Conference on VLSI Design, pp. 461–466 (2003)
35. Cong, J., et al.: High-Level Synthesis for FPGAs: From Prototyping to Deployment. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 30(4), 473–491 (2011)
36. Banerjee, P., et al.: A MATLAB Compiler for Distributed, Heterogeneous, Reconfigurable Computing Systems. In: Proceedings of IEEE Symposiumon Field-Programmable Custom Computing Machines, pp. 39–48 (2000)
37. MyHDL - From Python to Silicon: myhdl.org (May 2013)
38. Bellows, P., Hutchings, B.: JHDL - An HDL for Reconfigurable Systems. In: Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines, pp. 175–184 (1998)
39. Tripp, J.L., Jackson, P.A., Hutchings, B.L.: Sea Cucumber: A Synthesizing Compiler for FPGAs. In: Glesner, M., Zipf, P., Renovell, M. (eds.) FPL 2002. LNCS, vol. 2438, pp. 875–885. Springer, Heidelberg (2002)
40. Greaves, D., Singh, S.: Using C# Attributes to Describe Hardware Artefacts within kiwi, Specification, Verification and Design Languages. In: Forum on Specification, Verification and Design Languages, FDL 2008, pp. 239–240 (2008)
41. Luk, W., McKeever, S.: Pebble: A Language For Parametrised and Reconfigurable Hardware Design. In: Hartenstein, R.W., Keevallik, A. (eds.) FPL 1998. LNCS, vol. 1482, pp. 9–18. Springer, Heidelberg (1998)
42. Berry, G., Gonthier, G.: The Esterel Synchronous Programming Language: Design, Semantics, Implementation. Science of Computer Programming 19(2), 87–152 (1992)
43. BlueSpec (May 2013), `http://www.bluespec.com`
44. National Instruments LabVIEW (May 2013), `http://www.ni.com/labview/fpga`
45. FPGA Design and Codesign (May 2013), `http://www.mathworks.com/fpga-design`
46. Altium Designer (May 2013), `http://en.wikipedia.org/wiki/Altium_Designer`
47. Papakonstantinou, A., et al.: FCUDA: Enabling Efficient Compilation of CUDA Kernels onto FPGAs. In: Proceedings of the 7th IEEE Symposium on Application Specific Processors, SASP 2009, pp. 35–42 (2009)
48. khronos Group (May 2013), `http://www.khronos.org`
49. Owaida, M., Bellas, N., Daloukas, K., Antonopoulos, C.: Synthesis of Platform Architectures from OpenCL Programs. In: Proceedings of the 19th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 186–193 (2011)
50. Implementing FPGA Design with the OpenCL Standard (May 2013), `http://www.altera.com/literature/wp/wp-01173-opencl.pdf`