Automated HDL Generation: Comparative Evaluation

Yana Yankova, Koen Bertels, Stamatis Vassiliadis, Roel Meeuws, Arcilio Virginia
Computer Engineering Laboratory
Delft University of Technologies, The Netherlands
Email: {Y.D.Yankova, K.L.M.Bertels, S.Vassiliadis}@tudelft.nl, {rmeeuws, avirginia}@ce.et.tudelft.nl

***Abstract*— Reconfigurable computing (RC) systems, coupling general purpose processor with reconfigurable components, offer a lot of advantages. Nevertheless, currently a designer needs both in-depth software and hardware design knowledge to develop applications for such platforms. The automated hardware generation addresses this problem. However, the success of such tools remains marginal. This paper discusses the reasons for the lack of success. It presents a quantitative and qualitative comparison of three hardware generators using the following criteria: quality of the hardware model, the supported HLL constructs, and the level of automation.**

I. Introduction and Problem Description

Reconfigurable computing (RC) systems, coupling general purpose processors (GPPs) with reconfigurable components, offer a lot of advantages combining the flexibility of the software execution with the computational speed of the application-specific hardware. Nevertheless, currently a designer needs both in-depth software and hardware design knowledge to develop applications for such platforms. Therefore, dedicated tools and workbenches that make the hardware details transparent to the software designer are necessary. One component of a such workbench is an automated hardware design generator that takes its input from a high level language (HLL) such as C. In this generation process there are multiple challenges to be overcome. These challenges range from mapping the high-level (HL) constructs to the hardware to finding the proper set of optimizations and transformations that would lead to optimal hardware design. Multiple research projects, dating back more than 15 years, have addressed some of these problems. The goal of the first projects in this field was to answer the question whether it is possible to transform the high-level operations to hardware gates and wires? TransmogrifierC [1] is representative of this line of research. The goal of this compiler was to map simple integer arithmetic operations from C source to low-level gate logic. With the advance of the synthesis tools and the standardization of hardware description languages (HDLs), like VHDL, the focus shifted towards hardware implementation of more complicated high-level constructs and better exploitation of the hardware execution. SpC [2], for example, addressed the synthesis of pointers and dynamic memory management implementation. With the appearance of the RC systems, the research in the field branched with two main streams. Projects, like Handel-C [3], remained oriented towards the hardware designers. There, the C syntax is extended with constructs, exposing all hardware details to the designer. Other projects, like Streams-C [4] and SA-C [5], tried to hide the hardware details for the software designers. They used C variations, excluding problematic constructs and introducing language extensions that facilitated the optimizations and the hardware mapping. Those extensions however meant that existing applications had to be rewritten. Therefore, later projects like DEFACTO [6], SPARK [7] and ROCCC [8] considered unmodified C as input. Those projects emphasize parallelizing transformations and some also address memory access optimizations. In recent years, several commercial tools that generate hardware from HLL input also appeared (eg. Catapult-C [9], Impulse-C[10]).

Despite all those efforts, and for the following reasons, the automated hardware generation is yet to become a widely adopted industrial practice. (i) The first reason is the *limited subset* of the supported HLL constructs (respectively, *limited application domain*) for hardware implementation. The majority of the research projects targets predominantly applications characterized by perfectly nested loops and regular memory accesses. (ii) The second reason is the *lack of automation* of the generation process. Commercial tools do not limit the application domain but require extensive designer input in both the applied optimizations and the actual mapping process. Where the limited support for certain HLL constructs is less problematic, the lack of automation is more problematic as it implicitly requires hardware-design knowledge. (iii) The third and the most important reason is the *quality* of the generated designs, resulting in inefficient hardware utilization and low performance gain. This paper studies the above outlined problems through an empirical comparison between three research tools[1], namely ROCCC, SPARK and DWARV (a C to VHDL generator being developed as part of the Delft Workbench [11]). The comparison is performed at the three levels listed above. The data, used in the comparison, are obtained through generation and synthesis of VHDL models, based on C functions from different application domains.

The rest of the paper is organized as follows. Section II describes the evaluation criteria and the selected test applications. Brief overview of the evaluated tools is also included. Section III presents and discusses the obtained empirical results. Section IV outlines future research directions and concludes the paper.

II. Evaluation Setup

The comparison criteria are defined in the next subsection. The profile of the test applications is presented in Section II-C. The main features of the tested tools are discussed in Section II-B.

A. Criteria of Evaluation

The criteria used to evaluate the studied tools are grouped into three categories: supported HLL constructs, degree of automation, and quality of the generated designs.

In our experiments we consider combined GPP/FPGA co-execution with dynamic reconfiguration. Therefore, the evaluation of the designs **quality** is performed not only in terms of latency/throughput, but also in terms of utilized resources(area). The smaller designs would allow more than one function to reside simultaneously on the reconfigurable fabric. This brings two major benefits: (1) several algorithms can be executed in parallel; and (2) the reconfiguration frequency will be reduced, which will reduce the reconfiguration time and will improve the overall application performance. Nevertheless, there is a trade-off between the speed and the area of the designs. The

[1]Mentor Graphics currently does not provide academic licenses for Catapult-C; Impulse-C license was granted to the authors short before camera-ready deadline. Hence evaluation and comparison with a commercial tool was not possible

higher speed normally is achieved through parallel computation. The parallel computation implies higher resource utilization. Therefore, in order to properly evaluate the different tools, we need a metric that will capture objectively the size and the speed of the designs. To this purpose we propose to use throughput per slice[2].

Where the quality of the design can be expressed in numbers, quantitative metrics for the other two evaluation criteria cannot be defined. Nevertheless, they have direct impact on the tool appeal to the designer, hence they also have to be considered in the evaluation. Fully automated hardware generation is likely out of scope for the near future. But the degree to which human intervention is required to arrive at a particular hardware design does constitute a discriminative factor useful to compare different approaches. We will discuss the degree of **automation** in two aspects: (1) hardware resource specification; and (2) optimizations guidance. The description of the existing architecture (memory bandwidth, access time, etc) is not considered as resource specification as it does not vary between the designs. However, the number of adders to be used in a design is considered additional specification. The performance of given optimization is not considered as a guidance. However, deciding on which loop to perform unrolling and how many times, is considered an example of designer guidance.

The last criterion used in our evaluation is the supported **HLL constructs**. As all three tools take C source code as input, the discussion will be oriented towards the C language constructs and the degree to which those constructs are supported. This criterion is also hard to be expressed in quantitative terms. Moreover, some C constructs can be substituted by others while preserving the semantic (for example, a $do-while$ loop can be replaced with a for loop), hence support of a full-featured C is not a must from a semantic point of view. In addition, a definition of "minimum required" subset of constructs could be viewed as subjective. Therefore, we will evaluate the tools based on the lack of support of those constructs, used in our data set applications, for which in the given context its semantic counterpart cannot be found.

B. Tools Description

SPARK is designed to provide a toolbox for studying the effect of different optimizations on the area and delay of the derived designs. The emphasis in the project is operation scheduling techniques, aimed to increase the instruction-level parallelism in control-intensive kernels. SPARK has benefited of more than six years of research in the field of high-level synthesis.

ROCCC started three years ago and, as a descendant of SA-C and Streams-C, targets streaming applications. The emphasis in the project is to apply loop-parallelizing transformations and memory access optimizations. These optimizations are implemented assuming "sliding window" operations: the input stream is processed as sequence of smaller blocks (windows) that overlap with a certain offset.

DWARV is in its first year of development and currently does not target a particular application domain. It does not perform any optimization on the input code. The current emphasis of the tool is on straightforward generation of VHDL designs considering actual software/hardware co-execution on the MOLEN polymorphic processor [12], which involves a GPP augmented with reconfigurable accelerators.

[2]As we use this metric to compare the tools among each other and not the different models, generated by a single tool, we consider this metric applicable even for latency-driven designs

Every tool accepts as input C source code, possibly containing annotations. However, none of the tools supports full-featured C input. ROCCC, being strictly oriented towards streaming applications, accepts only perfectly nested constant-bound loops with an affine index function, operating on arrays. Moreover, all arrays are assumed to be located in the memory and no local data is allowed. SPARK, on the other hand, assumes that all data is transferred on the chip (either as local arrays or as input ports) before the computation starts and does not support any memory accesses. DWARV allows memory accesses and also supports local arrays, mapped to VHDL arrays. It does not impose limitations on the loop bounds. But it does not support control jumps. A more detailed list of the supported structures and the imposed restrictions is given in Section III.

C. Data Set

In order to compare the performance of C to VHDL translation tools, we have built a dataset of roughly 140 C functions. These functions come from a broad range of application domains, so we assume the results to be applicable to real world applications. As a consequence of the broad scope of the dataset, the candidate functions have varying types of algorithms and memory access patterns as can be observed in Table I. In the experiments, described in the next section, the original data sizes for the benchmarks were used.

Domain	bit-based	streaming	account-keeping[a]	control-intensive	integer-based	Memory-access
Compression	x		x	x	x	
Cryptography	x	x		x[b]	x	1D, Random
DSP	x	x		x[b]	x	1D, 2D
ECC	x	x		x	x	Static
Mathematics					x[b]	Static
Multimedia	x[b]	x		x	x[b]	Static, 1D, 2D
General Purpose			x[b]	x	x	1D, 2D, Random

[a]non-constant space-complexity
[b]only some instances in that domain express this characteristic

TABLE I
ALGORITHMIC CHARACTERISTICS OF APPLICATION DOMAIN.

As explained in the previous section each compiler has certain restrictions imposed on the C input code(see Table IV). Our intention was to modify the kernels such as to make them compliant with all three compilers. However, the combined restrictions of the three tools reduced the data set significantly. In order to have a broader set of data, we extended this subset with synthetic benchmarks specifically created to run on all tools. In Table II, the number of functions in each application domain for each tool under consideration is depicted. These numbers represent the number of kernels whose generated VHDL was successfully synthesized. There are some minor differences in the numbers of functions in the datasets for SPARK with and without optimizations, this is because some functions did not synthesized when SPARK was invoked without optimizations. As very few of the function matched the ROCCC's requirements, we were not able to prepare and present empirical data for the ROCCC compiler. The combined restrictions of DWARV and SPARK utlimately narrowed the test benchmark set to only 44 kernels.

III. Results

Automation: Table III summarizes the degree of automation of the tools.

- DWARV and ROCCC do not require any ***additional information***, besides the memory bandwidth and latency (DWARV)

Domain	Dataset	DWARV	SPARK
Compression	2	1	0
Cryptography	66	10	3
DSP	4	4	4
ECC	6	2	2
Mathematics	15	8	6
Multimedia	34	12	10
General Purpose	15	3	3
Synthetic	16	11	16
Total	**158**	**51**	**44**

TABLE II
FUNCTIONS PER APPLICATION DOMAIN.

in order to generate a design. However, SPARK does require additional specification of what kind (adders, multipliers, etc) and how many components to be used in the design. Such specifications give bigger control to an experienced designer, but will make it difficult for a non-expert to derive an optimal design in reasonable time. Moreover, if a type of resource, required by the C code, is not included in the resource description, VHDL design will not be generated.

- As DWARV currently does not perform any optimizations, the ***optimization*** criterion cannot be applied on it. ROCCC and SPARK, however, do perform optimizations. Nevertheless, both tools require designer guidance for performing them. For example, ROCCC requires external specification which loops should be unrolled and the unrolling factor. SPARK requires explicit specification of the type of code motions[3] and the scheduling strategies to be applied.

Compiler	Resource Specification	Optimizations Guidance
DWARV	No	N/A
ROCCC	No	Yes
SPARK	Yes	Yes

TABLE III
DEGREE OF AUTOMATION

High-Level Constructs: A summary of the supported C constructs is given in Table IV. The floating point types, the unions and the "address-of" operator are omitted from the table as they are not supported by any of the tools. The integer types and the compound statement are supported by all three tools, therefore they are also omitted from the table. Nevertheless, the sheer quantification of the supported C constructs cannot reflect the actual functionality, supported by the tools. As can be seen from Table V, the number of supported constructs by the different tools is almost equal. However, the limited support of the different variations of these high level constructs also explains the reduction by 50% of the kernels in our dataset. This is due to the fact that arithmetic and logical operations constitute the majority of the C constructs. The difference in the functionality is due to the context in which the control and memory accesses constructs as well as the compound data types are supported. DWARV is mainly restricted by the lack of support for structures. The most severe SPARK limitation is the requirement all arrays to be with constant size. This limitation stems from the fact that SPARK treats the arrays not as memory locations, rather as on-chip registers. Treating all storage elements as on-chip registers, prohibits also pointer arithmetic. The limitations imposed by ROCCC (for loops with regular stride, operating on arrays' windows, defined as an affine function of the loop iterator scalar) were even more severe, reducing the kernel dataset to 6% even though it does support a similar number of C-constructs.

[3]optimizations that speculatively move operations up and down on the control flow graph of the function in order to increase the instruction-level parallelism

Compiler	High Level Constructs	Supported Applications	Reason For Limited Support
DWARV	31/50	50%	Structures
ROCCC	31/50	6%	Perfectly Nested Loops
SPARK	37/50	50%	Constant size arrays

TABLE V
DEGREE OF AUTOMATION

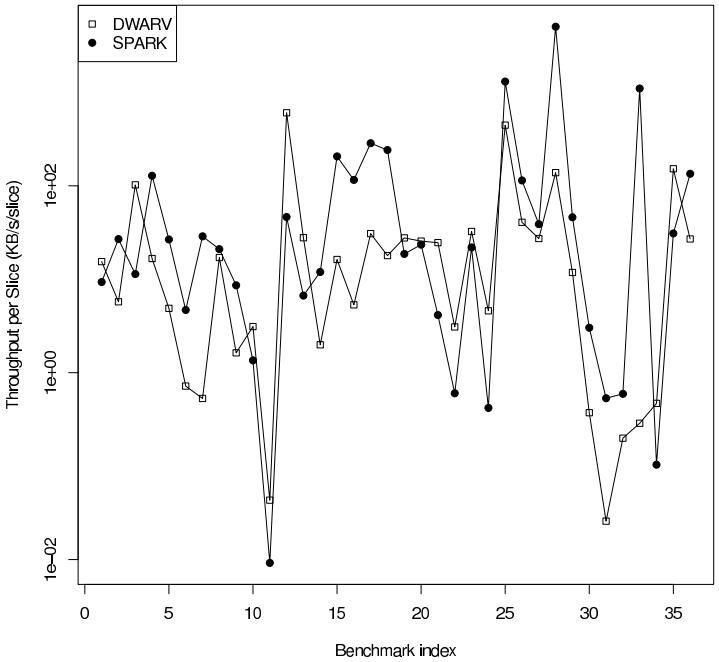

Fig. 1. Designs Quality

Quality: To evaluate and compare the quality of the generated designs, we have selected the metric throughput per slice. As explained above, only SPARK and DWARV were capable of successfully generating and synthesizing hardware. In order to assess the quality of the designs, we limit the comparison to the 44 kernels that could be synthesized by both tools. For the generation of the SPARK models, the default resources and optimizations settings were used. The actual synthesis was performed using the Xilinx ISE 8.2 tools for the Virtex II Pro. To estimate the latency (in number of cycles), the generated models were also simulated using ModelSim 6.1. The majority of the models, generated by SPARK, failed to simulate, because generates bidirectional ports for the in/out function parameters and the provided resolution function prevents proper input data feed. Therefore, to estimate the latency for those functions, we used the simulated latency of the DWARV's models and recovered the corresponding loop iteration counts. Based on those numbers and the number of states in the FSMs, the latency was computed. Then the estimated,

Compiler	Pointers	Aggregates	Arithmetic and Logic	Function Calls	Array Subscripting	Indirection	Labeled and Jump	Selection	Iteration
DWARV	As arrays	1D Arrays	Yes	No	Yes	Yes	No	if	for
ROCCC	As arrays	2D Arrays	excluding ? :	Inlined or lookup tables	Loop index	No	No	if	for[4]
SPARK V1.2	As function arguments	1D Arrays	excluding ? :	Inlined	Yes	No	$case$, $break$ in $switch$	Yes	for

TABLE IV
SUPPORT FOR C CONSTRUCTS

clock frequency and number of slices were used to compute the throughput per slice for each design. As can be seen in Figure 1. SPARK, outperforms DWARV for the majority of the functions. It is due to the fact that DWARV considers memory accesses, while SPARK assumes that all data reside on the chip in registers (either as IO ports or internal signals). Nevertheless, such an assumption has two major drawbacks. It connects directly the design size to the size of the input/output data. Small input/output data do not have great impact on the design area. However, 6 of the generated by SPARK functions require more than the available slices and 3 functions were too big to be synthesized at all. Moreover, in the context of heterogeneous systems, it is not reasonable to assume that each component in the system will process only autonomous data. The communication between the different components should not be ignored.

IV. Conclusion and Future Research

In this paper we presented a comparison between three VHDL generation tools and reported on various possibilities and limitations of each approach, using a set of criteria. The goal of this assessment was to evaluate the current state of the automated HDL generation when intended to be used by software designers, developing applications for RC systems. From automation point of view, DWARV requires least designer intervention, but it does not offer any optimizations in its current state. ROCCC hides the hardware details from the designer, but requires detailed guidance to perform high-level optimizations. SPARK has least automation among the three tools, requiring specification of the resources to be used and the optimizations to be applied. The three tools accept similar subset of HLL constructs, but the supported context and form differ. ROCCC is highly application oriented, which results in severe restrictions on the input code. SPARK and DWARV target broader range of applications, hence they have more relaxed restrictions compared to ROCCC. Nevertheless, none of the tools provided the necessary support in order all applications from the data set to be processed. Due to the severe restrictions by ROCCC, we had to limit our empirical comparison to only two of the tools. In terms of quality, SPARK, as expected, outperformed DWARV in the majority of the cases. The reasons of the poorer performance of DWARV lies in the lack of optimizations. Nevertheless, studying the different application profiles, it can be concluded that a single type of optimizations will not fit all application domains. For example, some of the applications are characterized with regular memory accesses and would benefit of data prefetching and computation pipelining optimizations. Other applications, however, are characterized with random memory accesses and cannot benefit from prefetching and pipelining optimizations. In other words, a C to VHDL compiler should not only apply certain set of optimizations. Rather it should be able to infer the semantic of the high-level code. Based on this semantic analysis, an appropriate computation model and optimization set should be selected. In our future research we aim to address this problem. A possible solution in this direction is the use of hardware design patterns, by analogy of the mainstream software design patterns where proven solutions are reused in new designs.

References

[1] D. Galloway, "The transmogrifier C hardware description language and compiler for FPGAs," in *IEEE Symposium on FPGAs for Custom Computing Machines*, 1995, pp. 136–144.

[2] L. Séméria and G. D. Micheli, "Resolution, optimization, and encoding of pointer variables for the behavioral synthesis from c." *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 20, no. 2, pp. 213–233, 2001.

[3] Handel-c language reference. [Online]. Available: http://www.celoxica.com/

[4] M. Gokhale, J. M. Stone, J. Arnold, and M. Kalinowski, "Stream-oriented fpga computing in the streams-c high level language." in *Proceedings of the 8th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2000)*, 2000, pp. 49–58.

[5] W. A. Najjar, A. P. W. Böhm, B. A. Draper, J. Hammes, R. Rinker, J. R. Beveridge, M. Chawathe, and C. Ross, "High-level language abstraction for reconfigurable computing." *IEEE Computer*, vol. 36, no. 8, pp. 63–69, 2003.

[6] P. C. Diniz, M. W. Hall, J. Park, B. So, and H. E. Ziegler, "Bridging the gap between compilation and synthesis in the defacto system." in *Proceedings of 14th International Workshop on Languages and Compilers for Parallel Computing (LCPC'01)*, 2001, pp. 52–70.

[7] Spark: A parallelizing approach to the high-level synthesis of digital circuits. [Online]. Available: http://mesl.ucsd.edu/spark/

[8] Riversite optimizing complier for configurable computing. [Online]. Available: http://www.cs.ucr.edu/ roccc/

[9] C-based design. [Online]. Available: http://www.mentor.com/products/c-based_design/

[10] Impulse-c. [Online]. Available: http://www.impulsec.com/

[11] Delft workbench. [Online]. Available: http://ce.et.tudelft.nl/DWB/

[12] Molen prototype. [Online]. Available: http://ce.et.tudelft.nl/MOLEN/Prototype/