An empirical comparison of ANSI-C to VHDL compilers: SPARK, ROCCC and DWARV

Arcilio J. Virginia, Yana D. Yankova, Koen L.M. Bertels
Computer Engineering Laboratory
Delft University of Technology, The Netherlands
Email: avirginia@ce.et.tudelft.nl, {Y.D.Yankova, K.L.M. Bertels}@tudelft.nl

Abstract— **Custom computing machines combine the flexibility of the general purpose processor with the high performance of the application tailored hardware. This combination results in a computing platform that allows for performance improvements to a wide range of applications. Nevertheless in order to develop applications for such a platform, a software designer currently needs also hardware design skills. In order to hide the hardware details from the designers and to assist in the non-trivial development process sevreral tools that translate ANSI-C to VHDL are proposed. In this paper we compare three such tools, SPARK, ROCCC and DWARV. The comparison is based on the performance of designs generated by the compilers and the experiences gained during the acquisition of empirical data.**

I. Introduction

Custom computing machines [1] are hybrid platforms which combine the benefits of both general purpose processors and reconfigurable components. The general purpose processor adds the flexibility in supporting a large class of applications and the reconfigurable components add application specific hardware acceleration. In order for an application to be mapped on such a hybrid platform it has to be partitioned to software and hardware segments. The software segments have to be further compiled for the general purpose processor. The hardware segments have to be translated into a hardware description language, such as VHDL. This translation is a non-trivial process and requires hardware design skills, if preformed manually. In order to relief the software designer from the requirement to gain hardware knowledge, the automation of this process is proposed. Nevertheless this automation presents other challenges.ANSI-C is an inherently sequential whereas hardware is fundamentally concurrent. The challenges in generating VHDL from ANSI-C lay in identifying and exploiting concurrency through compiler passes. Additionally ANSI-C contains code constructs not easily representable in hardware. The lack of expressive support for hardware design by ANSI-C has lead to compilers based on one of the following approaches: The first are hardware languages based on ANSI-C. One such example is Handel-C [2]. The ANSI-C code constructs make Handel-C easier to read as compared to VHDL. However, Handel-C is a semantically a new language and is geared towards hardware design. Streams-C [3] is an example of the second approach in which a subset of ANSI-C is extended with a library for identifying and supporting concurrency and specifying whether processes should run on the GPP or FPGA. The disadvantage of this approach is that existing ANSI-C applications would need to be rewritten. The focus of this paper is on the compilers adhering to the third approach. These compilers generate VHDL from unmodified ANSI-C. The three VHDL generators discussed in this paper are SPARK, ROCCC and DWARV. The aim of this paper is to compare these generators from a reconfigurable computing co-design environment context, as all three of them are intended as to be a part of such a tool chain. The three VHDL generators will be studied using qualitative and quantitative criteria. Through literature study and observations made during the acquisition of empirical data, we will compare the VHDL generation strategies of the tools. This includes documenting and comparing the supported ANSI-C subset and the restrictions the supported subset is subjected to. Hardware/software co-design environments accept ANSI-C as input. To make the C source compliant to the supported subset, the code may have to be rewritten. The effort required to do this, is investigated. The level of hardware knowledge required to derive the designs is determined. The generated designs are then synthesized to evaluate their performance. The rest of the paper is organized as follows. SPARK, ROCCC and DWARV will be briefly introduced in Section II. The criteria by which the VHDL generators will be compared is described in Section III. Results of the comparison are presented in Section IV. Section V the concludes the paper.

II. Tools Description

In this section we briefly introduce the ANSI-C to VHDL compilers which will be studied.

SPARK [4] is a VHDL generator designed as a research instrument, for investigating the effects of scheduling heuristics and compiler passes on control flow intensive ANSI-C functions. To this end the compiler features an intermediate representation [4], aimed at retaining the hierarchical structure of the source code, thus allowing both global and local optimizations through compiler transformations named code motions [5]. The aim is to generate quality VHDL designs, by identifying and exploiting concurrency throughout the intermediate representation.

ROCCC [6] is an optimizing compiler with the objective to generate hardware operations for acceleration on reconfigurable fabrics with onboard memory. Applications targeted by the compiler are streaming calculations involving loop and arrays. ROCCC aims at generating VHDL with good performance exploiting functional, loop and operation parallelism. In addition ROCCC aims at keeping the size of the generated designs to a minimum and employs a custom execution model to allow high computational throughput and reduce memory access latency through data reuse. The latter is accomplished through a windowing strategy in which Smart Buffers [7] are placed prior and after the data path.

DWARV [8], the Delft Workbench Automated Reconfigurable VHDL generator is part of the Delft Workbench tool chain [9]. The Delft Workbench is a hardware/software co-design environment targeting the MOLEN polymorphic processor [10] [11]. This tool chain accepts unmodified ANSI-C as input and compiles the application onto MOLEN. It identifies and selects candidate kernels (computation intensive functions) through profiling and resource estimation [12] prior to generating VHDL using DWARV. Unlike the SPARK and ROCCC, DWARV does not apply any optimizing transformations. Neither does the generator target a specific application domain. As a part of the Delft Workbench tool chain, DWARV was designed to support a large ANSI-C subset and generate a direct translation of the ANSI-C input into VHDL for execution on the MOLEN processor prototype [13].

III. Evaluation Methodology

In this section the quantitative and qualitative evaluation criteria are presented. A large data set of kernels was gathered to perform the comparison. In this section this data set is also presented.[10], [11]

A. Comparison Criteria

There are five **qualitative** aspects of the generators, which are considered. The first two are the *supported ANSI-C subset* and *restrictions* posed on this subset. The third aspect is the *required rewriting effort* exerted to make existing ANSI-C code compliant to the generators subset. The *knowledge of hardware* required to generate synthesizable VHDL using SPARK, ROCCC and DWARV is the fourth qualitative aspect. And the final qualitative criteria are the *testability and readability* of the generated VHDL designs. These qualitative measures are of importance to the designer. They describe the level of (hardware) knowledge and manual intervention is needed by the designer to generate and validate designs. From the perspective of a hardware/software co-design environment a VHDL generator should preferably be fully automated. By automation is meant the generation of VHDL from an unmodified ANSI-C file without user intervention. None of the three approaches discussed here are fully automated. The qualitative criteria are determined through literature study and during experimentation. The supported subset is expressed in percentages, data types, operators and control flow code constructs. Except for supported subset the other qualitative criteria are not quantifiable by numbers.

Quantitative measures used in studying the generator approaches are acquired through synthesis and simulation of the designs generated by the three tools. From these measures the quality criterion throughput/slice was calculated. It is a measure that relates the speed of a design to the utilized area and can indicate how efficient the performance of a design is. To acquire the empirical data needed to calculate the throughput/slice, the generated designs are synthesized and simulated. The tools used are Xilinx ISE 8.2 and ModelSim. The synthesizer was set to target a Xilinx Virtex II Pro FPGA.

B. Data Set

To compare the generators using the throughput/slice, a large data set consisting of ANSI-C functions was gathered. These functions stem from seven separate application domains and are complemented with several synthetic functions. The first two columns of Table I present the domain and the number of functions considered for the study. These functions consist of data, control, and data and control intensive kernels. Data intensive functions heavily read and write to memory. Whereas in control intensive functions conditional evaluations are clearly dominant; and in case of data and control intensive, such functions constitutes the load of the data transfer and the conditional control flow. In order to compare

TABLE I
Number and domain of the data set functions.

Application Domain	**Considered**	**Made compatible**	**Synthesized**
Compression	2	0	0
Cryptography	66	5	3
DSP	4	5	5
ECC	6	2	2
Mathematics	15	6	6
Multimedia	34	11	10
Miscellaneous	15	2	2
Synthetic	16	14	5
Total	158	45	33

SPARK, ROCCC and DWARV, the input functions had to be rewritten. Initially the functions were to be rewritten conform to the combined requirements of all three generators. Each approach supports its own subset of ANSI-C. However, the requirements of ROCCC are too stringent and its unavailability at the time of experimentation led to the functions being rewritten to be compliant to SPARK and DWARV.

In the following section we will discuss the results of the qualitative and quantitative analysis.

IV. Results

Here we present the qualitative and quantitative results of our study into the VHDL generators SPARK, ROCCC and DWARV. The first set of results presented are the qualitative. These results stem from the experiences gained during the acquisition of the empirical results for the quantitative comparison.

A. Qualitative Analysis

Supported ANSI-C Subset: The subset of ANSI-C supported by the VHDL generators is expressed in percentages and presented in Table II. The data types consist of integer, floating point, pointer and aggregate types such as arrays, structures, unions and enumeration. All three generator approaches support integer types. None of the tools support floating point, structures and unions. DWARV supports the largest percentage of types, due to enumeration and pointers. These two data types are not fully supported by SPARK, where pointers are treated as arrays. Of the

TABLE II
Percentage of support for main ANSI-C features.

	DWARV	**ROCCC**	**SPARK**
Data types(13)	69%	54%	54%
Operators(17)	85%	62%	62%
Flow of control(9)	22%	33%	56%
Total	63%	51%	54%

operators none of the tools support field selection and address of object operators. DWARV includes indirection via pointers, modulus and conditional expressions, which are operators not supported by SPARK and ROCCC. Therefore DWARV supports the largest set of operators. In case of both data types and operators ROCCC and SPARK support the same types.

None of the generator approaches, support break, continue and goto control flow constructs. The ANSI-C subset of SPARK includes, in addition to for- and if-statements, while-, do- and switch-statements. Whereas, DWARV and ROCCC support only for- and if-statements. In case of the latter approach, labels are additionally supported. These are required to identify loops in functions for generating designs.

Code Restrictions: Restrictions are posed by the generators on the supported subsets. These restrictions stem from the design strategy of the compilers. ROCCC poses the most restrictions due to its streaming application strategy. It requires for-loops to be perfectly nested and there must be an input and output array part of the loop body. The indices of these arrays must be equal to the loop counter and the number of loop iterations has to be equal to the number of output array elements.

SPARK stores data locally on chip. One consequence of this strategy is no support for pointers arithmetic. Additionally all arrays must be global, have a fixed size and the index may not be negative.

DWARV is the least restrictive of the generators. SPARK and ROCCC are designed to target control intensive and data intensive kernels respectively, whereas DWARV design is not focused on one type of kernel. Therefore it poses the lightest set of restrictions.

Rewriting Effort: The rewriting effort is dependent on the style in which the data set functions are written, the supported subset and severity of the restrictions. Here we do not consider the effort exerted for rewriting functions using structures, floating point and unions. This effort is equal for all three tools.

ROCCC requires the most effort in rewriting code. The generator has strict requirements concerning for-loop header and body. The effort required by SPARK is moderate due to its lack of support for pointers and requirements surrounding arrays. DWARV supports both pointer and pointer arithmetic and does not pose severe restrictions; therefore rewriting code for this generator requires the least amount of effort.

Readability and Testability of Designs: The readability and testability of the designs generated by DWARV and SPARK are important during functional verification. DWARV and SPARK both generate a finite state machine (FSM) and staged execution model. SPARK designs resemble the C source. This is achieved by reusing the original C source variable names and through the use of a proprietary SPARK VHDL type, *WiredOrInt*, to allow integer arithmetic. Thus allowing e.g. arrays to be used as in ANSI-C. This makes the SPARK FSM easier to read. However, SPARK uses if-statements in its designs, whereas DWARV uses case-statements. The advantage of this is that as the design grows in size, the FSM written in case-statements is easier to read. To verify the validity of the generated designs they need to be simulated. In comparison to SPARK, DWARV offers better support for designs simulation. The former requires a new test bench for each design. This is again a consequence of storing data locally on chip. The input and output ports vary per design. DWARV generated designs have a custom input/output interface compliant with the MOLEN CCU [13]. As a result only one test bench is required which can read and write input and results to files.

Required Hardware Knowledge: The required hardware knowledge is dependent on the level of control offered by the compiler to the designer, over the generation of the VHDL designs. This influence is exerted by the designer through choosing optimizing compiler passes such as loop unrolling, which influences the size of the design. ROCCC focuses on generating optimized VHDL from ANSI-C for-loops. The compiler offers its designer 4 choices in compiler transformations aimed at for-loops. These transformations affect the speed and size of the final design. SPARK offers designers the most control over their designs. Through 14 compiler transformations, compilation options and by specifying which resources are available to be scheduled. These options make SPARK a powerful tool for creating an efficient design. However, this level of control over the final design in combination with having to write a test bench for each new design, makes SPARK the compiler requiring the most hardware knowledge. DWARV currently requires the least hardware knowledge. It does not have optimizing transformations to offer designers and similar to ROCCC, no resources have to be specified. In Table III a short summary is presented on the above discussed comparison results.

TABLE III
SUMMARY QUALITATIVE COMPARISON.

	DWARV	**ROCCC**	**SPARK**
Largest supported ANSI-C subset	**1**	3	2
Most restricted ANSI-C subset	**3**	1	2
Highest rewriting effort required	**3**	1	2
Best readable designs	2	–	**1**
Testability of designs	**1**	–	2
Most hardware knowledge required	**3**	2	1

Explanation : In this table the number represent the rank of the compiler given the criteria. When made **bold**, this number indicates the best performing compiler. In two instances ROCCC could not be tested which is indicated by –.

B. Quantitative Analysis

Our study continues with the quantitative analysis in which we report the reduction of the data set, the simulation results and performance comparison of SPARK and DWARV generated designs.

Reduced data set: The data set of functions was reduced by the combined restrictions of both tools. Here the generator with the most stringent requirements dominated. For each application domain the number of kernels made compliant and kernels synthesized are presented in Table I. Ultimately only 33 kernels could be synthesized. Both DWARV and SPARK are research tools. However, unlike DWARV, SPARK is not being actively developed. DWARV does not perform optimizing transformations and has less stringent requirements as compared to SPARK. This in combination with testing DWARV with a large data set has lead to the tool generating more synthesizable VHDL code. For generating designs SPARK was configured using the default resource file and compilation options. Nevertheless, it does not always generate synthesizable VHDL or synthesizable VHDL that can be simulated.

Simulation Results: In Table IV a summary is presented on SPARK simulation results. Sixteen designs are successfully simulated and half of the designs required minor modifications in order to simulate. In seven cases SPARK generated incorrect designs. In these designs the array index would overflow. Whereas in the original ANSI-C source code this was not the case. Seventeen SPARK generated

TABLE IV
Number of simulated and failed SPARK kernels.

	Simulated	Failed	Error
Modified	8	7	Array indexing
Unmodified	8	17	I/O
Total	16	24	

designs could not be simulated. These designs have bidirectional ports of the SPARK proprietary type *WiredOrInt*. Of which the resolution function does not operate correctly. Prior to storing input values on chip, a lower bound integer value specified in the SPARK resource file, is added to the incoming value. Attempts to remove the initial value or write to these bidirectional ports was in vain. In case of DWARV, all designs were simulated. As explained earlier, at the time of data acquisition the generator was under-development and at any stage of the design VHDL generation and validation errors could be corrected in the compiler. The ANSI-C kernel inputs and outputs were recorded and used in the simulation to verify the correct operation of the generated designs. In case of the seventeen SPARK designs, they were presumed to be functioning correctly.

Performance Comparison: In order to calculate the throughput/slice (1), the following measures are acquired: number of slices, delay (period) and number of cycles required by the design to produce a result. The latter measure was required to calculate the latency (2) and was computed by dividing the time it had taken the design to produce results, by the simulated clock period.

$$throughputperslice(KB/sec/slice) = \frac{\frac{(numberofbytes)*10^9}{latency*1024}}{slices}; \quad (1)$$

$$Latency(nanoseconds) = delay * cycles; \quad (2)$$

For the seventeen SPARK designs, their latency was estimated by calculating the loop iteration counts using the DWARV simulated latencies. By using the previously calculated value and the number of states in the FSM the latency for SPARK designs was estimated.

The results of the comparison are illustrated by category kernel in figures 1, 2 and 3. Each generated design pair corresponding to one kernel has been normalized to the highest of the two values. As expected the majority of SPARK designs outperform DWARV, especially in case of control intensive designs. This can be attributed to the *number of cycles*, *data storage strategy* and *optimizing compiler transformations* applied. **Cycles** are the total number of times the FSM states are traversed. The more states used to perform the functions of a loop, the larger the number of cycles. DWARV designs require more cycles, due to the fact that DWARV reads and writes data to and from memory. When comparing the number of cycles of the 16 simulated SPARK designs to the DWARV designs, on average 56% more cycles are required by DWARV.

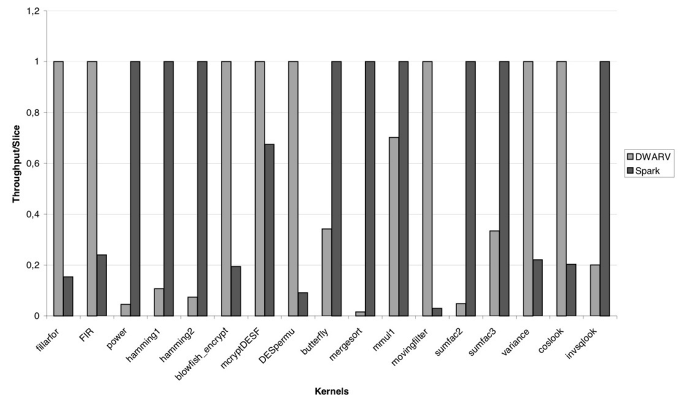

Fig. 1. The throughput/slice of the data intense designs.

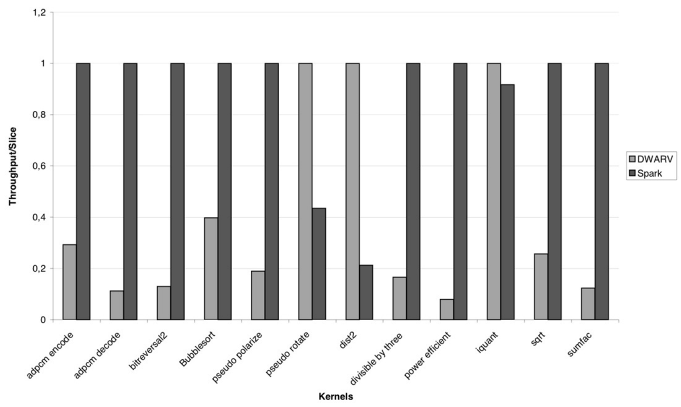

Fig. 2. The throughput/slice of the control intense designs.

All data in SPARK designs are stored on chip. This reduces the number of cycles required by the design and translates into an advantage in lower latency and higher throughput. Only 4 of the 33 kernels had a higher latency as compared to DWARV designs and of these 4 designs, the number of cycles are calculated for 3 because they did not simulate and in case of the fourth kernel the scheduler did not have enough resources to fully exploit concurrency. The disadvantage of storing **data** on chip is that the size of the design grows dependent on not only the number of scheduled resources but also on the size of the input. This can diminish the advantage of having data stored locally. In comparison to SPARK, DWARV designs

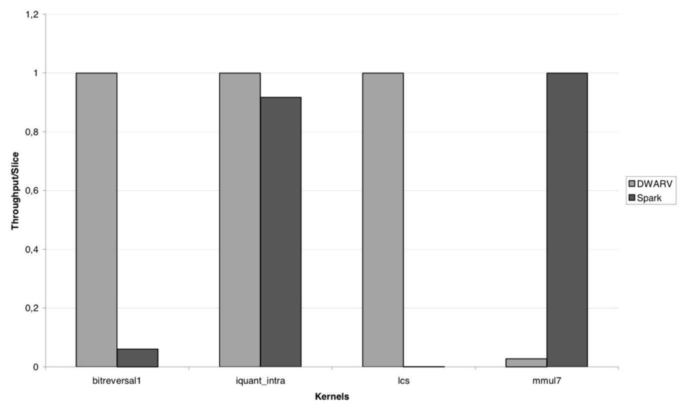

Fig. 3. The throughput/slice of the data and control intense designs.

TABLE V
Influence of data increase on synthesis results SPARK.

Range	Slices	FlipFlops	LUTs	IOs
range 1: 2^{32}	195	296	209	547
range 2: 2^{16}	127	207	129	292
range 3: 2^{8}	83	134	85	156
unsigned 2^{4}	62	98	68	88

may not have the best performance, nevertheless they require overall less area on the FPGA. In Table V a numerical example is presented consisting of synthesis results whereby the range of an integer in the SPARK resource file is constantly increased after each synthesis. The function used is an algorithm for multiplying three 2 by 2 matrices concurrently. Four integer arrays are used, one as output and three input. The input of the function consisted of 4 integer arrays and one argument. A reduction of 35% of the area can be observed when the integer range is reduced from *1* to *2* and from the latter to range *3*; the last reduction is of approximately 25%. Out of the 33 kernels all the DWARV designs fit on the FPGA, in 6 cases SPARK designs use more resources than available on the FPGA.

SPARK designs outperform DWARV due to in part the latter lack of **optimizing transformations**. The optimizing transformations applied by SPARK are speculation, reverse speculation, early conditional execution and conditional speculation. These transformations exploit concurrency by moving operations across control structures. The result of these moves are less states in the execution model as the longest path is reduced. DWARV performs no transformations that optimize the design. The optimizing transformations of SPARK are specifically applicable to control intensive kernels. This can explain why the largest group of SPARK designs outperforming DWARV are control intensive. Two of the three SPARK designs which are outperformed by DWARV are due to the large area they required. The third of the control intensive kernels can be explained by the number of cycles being calculated as worst case.

In this section the results have been presented of the qualitative and quantitative comparison of SPARK, ROCCC and DWARV. Of the three compilers only SPARK and DWARV generated synthesizable VHDL.

V. Conclusion

In this paper we presented our findings of the qualitative and quantitative comparison of DWARV, ROCCC and SPARK. Our goal was to asses these generators from a reconfigurable computing co-design environment context. In this context the designer is assumed to be a software engineer with limited hardware knowledge. The qualitative comparison reveals DWARV to support the largest subset of ANSI-C and poses the least restrictions on this subset. Resulting in lower required effort when rewriting functions as compared to the other two generators. This generator also demands the least hardware knowledge. DWARV currently has no optimizing transformations that can be applied by the designer and one test bench suffices for all generated designs. ROCCC posses the most stringent requirements on its supported subset and therefore requires the most effort when rewriting code, due to its windowing strategy. Although as compared to SPARK far less hardware knowledge is required. SPARK demands the most designer input and hardware knowledge. This generator offers the designer the most control over the design and for each generated design a separate test bench has to be written for design verification. Through the quantitative analysis we have established that there is a direct correlation between the severity of the restrictions on the ANSI-C subset, the rewriting effort, and the number of supported kernels. Furthermore, the successful generation of VHDL designs is by no means a guarantee that the designs can be successfully simulated or that they are functionally correct. SPARK outperformed DWARV in the majority of the cases. This is due to the fact that DWARV does not have optimizing transformations and does not store data on the FPGA. Regardless of the fact, not all designs generated by SPARK could benefit from the optimizing transformations because these are specifically targeted at control intensive kernels.

References

[1] M. Sima, S. Vassiliadis, S.D. Cotofana, J.T.J. van Eijndhoven, and K.A. Vissers *"Field-Programmable Custom Computing Machines - A Taxonomy"*, Proceedings of the 12th International Conference on Field-Programmable Logic and Applications (FPL 2002) September, 2002.

[2] *Handel-C language reference manual*, Celoxica Limited, 2004.

[3] M. Gokhale, J.M. August 2001, 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.

[4] R.K. Gupta, A. Nicolau, S. Gupta and N.D. Dutt, *"SPARK: A high-level synthesis framework for applying parallelizing compiler transformations."*, Int. Conference on VLSI Design 2003, January 2003

[5] S. Gupta, N. Savoiu, N.D. Dutt, R.K. Gupta and A. Nicolau, *"Using global code motions to improve the quality of results for high-level synthesis."*, IEEE transactions on computer-aided design of integrated circuits and systems, February, 2004

[6] W. Najjar, B.A. Buyukkurt and Z. Guo, *"Compiler optimization for configurable accelerators"*, Int. Workshop On applied Reconfigurable Computing (ARC 2006), Delft, The Netherlands, 2006.

[7] B.A. Buyukkurt, Z. Guo and W. Najjar*"Input data reuse in compiling window operations onto reconfigurable hardware."*, Proc. ACM Symp. On Languages, Compilers and Tools for Embedded Systems (LCTES 2004), Washington DC, June 2004.

[8] Y. D. Yankova, G.K. Kuzmanov, K.L.M. Bertels, G.N. Gaydadjiev, J. Lu and S. Vassiliadis, *"DWARV: DelftWorkbench Automated Reconfigurable VHDL Generator"*, In Proceedings of the 17th International Conference on Field Programmable Logic and Applications (FPL07), Delft, The Netherlands, 2007.

[9] K.L.M. Bertels, S. Vassiliadis, E. Moscu Panainte, Y.D. Yankova, C.G. Galuzzi, R. Chaves and G.K. Kuzmanov, *"Developing applications for polymorphic processors: the delft workbench"*, Delft, The Netherlands, 2006.

[10] S. Vassiliadis, S. Wong and S.D. Cotofana *"The MOLEN $\rho\mu$ -coded processor"*, in the International Conference on Field-Programmable Logic and Applications (FPL), Springer-verslag Lecture Notes in Computer Science (LNCS) vol.2147, pp 275-285, August 2001.

[11] S. Vassiliadis, S. Wong, G.N. Gaydadjiev, K. Bertels, G. Kuzmanov and E.M. Panainte*"The MOLEN Polymorphic processor"*, IEEE transactions on Computers, pp. 1363-1375, November 2004.

[12] R.J. Meeuws, Y.D. Yankova, K.L.M. Bertels, G.N. Gaydadjiev and S. Vassiliadis *"A Quantitative Prediction Model for Hardware/Software Partitioning"*, Proceedings of 17th International Conference on Field Programmable Logic and Applications (FPL07) August, 2007.

[13] G.K. Kuzmanov, G.N. Gaydadjiev and S. Vassiliadis*"The MOLEN Processor Prototype"*, Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2004), pp. 296-299, April 2004.