

A quantitative evaluation of C-based synthesis on heterogeneous embedded systems design

Omar Hammami, Zoukoun Wang, Virginie Fresse, Dominique Houzet

▶ **To cite this version:**

Omar Hammami, Zoukoun Wang, Virginie Fresse, Dominique Houzet. A quantitative evaluation of C-based synthesis on heterogeneous embedded systems design. IEEE-ISCAS2008 Conference, Jun 2008, Seattle, United States. pp.368-371, 10.1109/ISCAS.2008.4541431 . hal-00322712

HAL Id: hal-00322712
https://hal.science/hal-00322712

Submitted on 18 Sep 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Quantitative Evaluation of C-Based Synthesis on Heterogeneous Embedded Systems Design

Omar Hammami, Zoukun Wang
ENSTA, 32 Boulevard Victor,
75739 Paris, France

Virginie Fresse
Hubert Curien Laboratory, 18 rue
Benoit Lauras 42000 Saint
Etienne, France

Dominique Houzet
GIPSA Lab, INPG, 46 avenue
Félix Viallet, 38031 Grenoble,
France

Abstract- C-based design techniques and methodologies have been proposed to tackle the complexity of heterogeneous embedded systems. The heterogeneity comes in the functionalities and the implementation requirements. Various IPs with diverse complexities and functionalities can be selected to build an heterogeneous system. However, implementation hints should be available at the highest possible level of abstraction. In this paper, we conduct a quantitative evaluation of C-based design of heterogeneous embedded systems and point out the impact of behavioral synthesis on partitioning.

I. INTRODUCTION

Embedded systems are increasingly complex and heterogeneous in their implementations, their functionalities and their usages [1-2]. Research efforts have focused on specifications and design methodologies based on Models of Computations (MoC) but little work have been done so far on implementations issues and their impact on heterogeneous embedded systems design methods. Due to the high level of abstraction required for the specification of heterogeneous embedded systems implementation issues should be tackled as well at the highest possible abstraction level and naturally through behavioural synthesis or estimate. This being done in C-based framework the question at hand is the impact of C-based behavioural synthesis on C-based specification and modelling framework for heterogeneous embedded systems.

The heterogeneity of heterogeneous systems mainly takes its source in the architecture components and the model of computations used in the specification of the systems. However, this heterogeneity could advantageously be enriched by additional attributes which contribute to the "heterogeneity" of a system such as physical implementation details. Research conducted in heterogeneous systems (e.g. [3-9].) tackle component-based framework for heterogeneous modelling with the objective of providing a modelling and simulation environment, SystemC based extensions and heterogeneous specification methodologies (HetSC). Horizontal heterogeneity and vertical heterogeneity have been defined [3] for respectively defining the ability to support several models of computations and the ability to support evolution among models of computations along the design process. The first is essential in defining the syntax and semantics of the specification and the set of rules to build a heterogeneous specification. This requires addressing theoretical aspects in concurrency and communication semantics.

We are interested in this paper by vertical heterogeneity through high level synthesis of components expressed with C-based languages. Independently of heterogeneous modelling and specification these recent years have seen the emergence of c-based synthesis tools. The issue at hand is how vertical heterogeneity driven by c-based high level synthesis tools may affect horizontal heterogeneity ? In the first step of the design flow horizontal heterogeneity mainly concerns concurrency and functionality. However, annotation and back annotation of physical constraints in terms of area, power and floorplan may positively contribute in large scale SOC where GALS models dominate. This is even mandatory in resources constrained technologies such as FPGA where embedded RAM (block RAM – BRAM), hard DSP blocks and multipliers are clearly specified as part of the device. In order to keep the overall heterogeneous specification and design process fast this annotation should be based on high level synthesis in the same system-level specification language. Currently SystemC is the most widely language for this purpose. The second step operates a synthesis from a C-based (ANSI C, synthesizable subset of SystemC) description into VHDL. This step operates a transformation where the concurrency and communication semantics are expected to be preserved and not modified. Step 3 will take the resulting VHDL to operate physical synthesis in a classic way onto a target technology.

However VHDL synthesis does not consider MOC and all high level concepts are unknown at this level. Furthermore in resources bounded devices such as FPGA the place and route may have numerous solutions in terms of area and frequencies. For example, the vertical heterogeneity of two communicating clocked synchronous MOCs may result in a GALS model due to large discrepancies between the respective working frequencies or floorplan constraints may place each clocked synchronous MOC sufficiently far away on the SOC to justify again with the same consequence a GALS model. With this situation in mind it is essential to quantify

978-1-4244-1684-4/08/$25.00 ©2008 IEEE

the variations in C-based high level synthesis tools to identify the type of annotations we need.

II. C-BASED SYNTHESIS

Familiarity is the main reason C-like languages have been proposed for hardware synthesis. Another common motivation is HW/SW co-design. Using a single language for HW/SW designs simplifies the migration task and ensures an entire system verification. Important uses of a design language in addition to synthesis are validation and algorithm exploration (including an efficient partitioning). The C-language has no support for parallelism and as a consequence, either the synthesis tool is responsible for finding it or the designer is forced to modify the algorithm and insert explicit parallelism. The C-language is also mute on the subject of time. Data types are another central difference between hardware and software languages. All these characteristics must be considered when designing C-like hardware languages [1]. All characteristics related to the considered tools are analyzed in the rest of the paper. We selected 3 commercial tools, presented in Table I [10-12].

TABLE I. C-BASED ENVIRONMENTS

Tool	Compagnies
ImpulseC	Impulse
Handel-C	Celoxica
Agility SystemC Compiler	Celoxica

III. DESIGN CASES

In order to evaluate the synthesis efficiency of the previously described tools the use of commonly accepted benchmarks for c-based synthesis would have been useful. However, so far no benchmarks have been released from the OSCI Synthesis Working Group which defined the synthesizable subset of SystemC nor by any other body. We decided to select our own case studies composed of short and simple functions in order to allow reproducibility. The selected cases are a 3x3 mean filter, a 3x3 median filter and a FFT.

The filtering benchmarks are based on three 32-bit streaming inputs providing the pixels (bytes) 4 by 4 from three consecutive lines of the image to be filtered and produce a line of pixels of the resulting image. The size of the internal storage is 6 * 3 pixels to produce 4 pixels at a time. The last benchmark is the radix-4 FFT on 256 complex values (16-bit).

Three solutions are implemented and evaluated. The first one is a sequential one with RAM as internal storage. The second one is a parallel/pipeline solution with RAM as internal storage. Three separate RAMs are used to allow parallelism between the three inputs. The third solution is a parallel/pipeline solution with registers as internal storage.

IV. RESULTS

In the framework of this study we selected the Xilinx Virtex-4 technology as the target technology [13]. Physical synthesis have been conducted using Xilinx tools with an automatic exploration of options spanning a wide range from area oriented towards speed oriented with optimization effort, density factor varied at the different steps. This comes as a complement to optimization techniques employed by C-based synthesis such as for example speculative execution. The mutual effects - potentially inhibitory - of C-based synthesis followed by VHDL physical synthesis are unspecified in any of the tools documentation.

A. Performance results

The performance results are obtained for each IP with the different tools. Our metrics are clock frequency, latency and cycle per result.

TABLE II. IMPULSEC PERFORMANCE RESULTS

	Cycle/results	Latency	Clock period
Mean pipe reg	2	13	4.5
Mean pipe RAM	12	14	12
Mean sequential RAM	14	14	15
Median pipe Reg	2	3	10
Median pipe RAM	12	24	7.9
Median sequential RAM	333	333	11
FFT 256 sequential RAM	30 720	30 720	19
FFT 256 pipe RAM	1	7	8.5

TABLE III. HANDEL-C PERFORMANCE RESULTS

	Cycle/results	Latency	Clock period
Mean pipe reg	10	10	5
Mean pipe RAM	18	18	6.2
Mean sequential RAM	65	65	5.8
Median pipe Reg	16	16	5.5
Median pipe RAM	26	26	7.3
Median sequential RAM	438	438	7.5
FFT 256 sequential RAM	22553	22553	8
FFT 256 pipe RAM	2630	2630	8.5

TABLE IV. AGILITY PERFORMANCE RESULTS

	Cycle/results	Latency	Clock period
Mean pipe reg	2	12	5.246
Mean pipe RAM	4	36	6.047
Mean sequential RAM	9	9	5.130
Median pipe Reg	2	14	5.221
Median pipe RAM	4	38	6.035
Median sequential RAM	11	11	4.601
FFT 256 sequential RAM	2	5504	12.048
FFT 256 pipe RAM	2	12	5.246

The variability of results between the tools comes from different reasons. Firstly, the RAM implementation is a direct implementation with no multiplexing of resources: here the three RAM of the filters are accessed with one access per clock cycle resulting in a limitation of the pipeline rate of

twelve cycles per data produced. Secondly the analyzis of the results can be divided in two parts: first the SystemC and Handel-C tools which need to explicitly program the pipeline and second the ImpulseC tool where the C-code is functional with no specific programming. Also, Impulse-C is timing constrained. The number of stages of the pipeline is not precisely controlled as it is the case with SystemC and Handel-C but undirectely through their constraints. The automatic exploration of different options and constraints is the only solution to obtain the best compromised between the different constraints as the impact of the rate/latency of the pipeline on the area/frequency is not straightforward. The difference of rate between a pure sequential solution and a fully pipeline solution can be more than two orders of magnitude. This is the main source of performance/area trade off at this level. This difference is amplified with the implementation variability which is masked in these results.

B. Area performance

The area results have been obtained through VHDL generation of the various case studies followed by synthesis and place and route using Xilinx XST tool. Our area metrics are composed of the various resources present in the Xilinx Virtex-4 that is slices, DSP, BRAM.

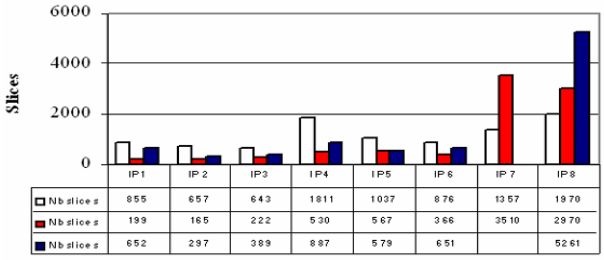


Figure 1. Resources (number of slices) for all tested IPs with 3 different Place and Route options.

It should be reminded that obviously synthesis and place and route can incur large variations if no constraints are imposed and large chips are selected. In our case we allowed the tools to synthesize and place and route without constraints in the first step and then followed this step with a constrained place and route. Results achieved are superior with the constraints. We applied an automatic exploration of synthesis and place and route options for each case study. The clock periods variations in Fig 2-3 are obtained with a variation of area cost between 50 and 100 slices.

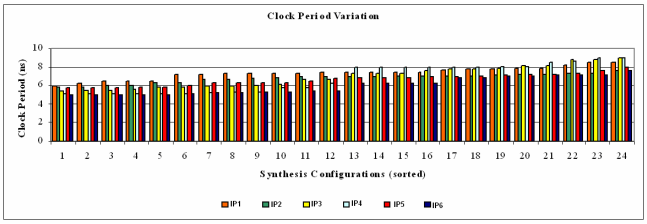


Figure 2. Agility - Xilinx XST VHDL Synthesis tool variation

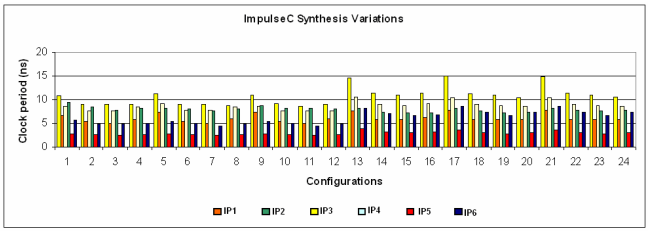


Figure 3. ImpulseC - Xilinx XST VHDL Synthesis tool variation

C-based synthesis may generate very different implementations resulting from C-based. It should be noted that with heterogeneous devices such as Virtex-4 where hard cores are embedded the place and route tools may decide to implement a function in the vicinity of such embedded cores even if no interaction exists. This affects the quality of the implementation as the logic is spread out on the circuit. This clearly shows that there is a missing link between the MoC based modeling of heterogeneous systems and the physical implementation. A feedback is necessary to help vertical heterogeneity.

TABLE V. IMPULSEC PERFORMANCE RESULTS

	Nb slices	Nb LUT	Nb FF	Nb LUT RAM	Nb BRAM
Mean pipe reg	855	1278	1153		
Mean pipe RAM	657	476	485	96	
Mean sequential RAM	643	874	162	96	
Median pipe Reg	1811	3533	148		
Median pipe RAM	1037	1351	555	224	
Median sequential RAM	876	1374	352	224	
FFT 256 sequential RAM	1357	2220	606		7
FFT 256 pipe RAM	1970	2463	1211		7

TABLE VI. HANDEL-C PERFORMANCE RESULTS

	Nb slices	Nb LUT	Nb FF	Nb LUT RAM	Nb slices
Mean pipe reg	199	235	178	48	199
Mean pipe RAM	165	202	163	48	165
Mean sequential RAM	222	275	192	48	222
Median pipe Reg	530	636	307		530
Median pipe RAM	567	960	242	48	567
Median sequential RAM	366	370	248	48	366
FFT 256 sequential RAM	3510	6992	647		3510
FFT 256 pipe RAM	2970	4463	1022		2970

TABLE VII. AGILITY PERFORMANCE RESULTS

	Nb slices	Nb LUT	Nb FF	Nb LUT RAM	Nb slices
Mean pipe reg	652	981	655		652
Mean pipe RAM	297	217	423	24	297
Mean sequential RAM	389	526	386		389
Median pipe Reg	887	1297	965		887
Median pipe RAM	579	627	685	24	579
Median sequential RAM	651	1151	555		651
FFT 256 sequential RAM	5261	7380	5094	512	5261
FFT 256 pipe RAM	652	981	655		652

V. DISCUSSION: IMPACT OF THE RESULTS ON C-BASED DESIGN

The C-to-hardware compilers considered here take two approaches to concurrency. The first approach chosen by Handel-C and SystemC adds parallel constructs to the language. It forces the programmer to expose most concurrency that is not a difficult task in major cases. Handel-C provides specific constructs that dispatch collections of instructions in parallel. These additional statement constructs can be used by any programmer. For all the implemented filters, adding manually parallelism is an easily task that can be achieved by any programmer. On the other hand, pipeline extraction can become a tricky task as algorithm must be written in that way. An example was the FFT algorithm implementation: adding pipeline from a sequential code can take a long time and changes are important to make. The other approach lets the compiler identify parallelism helped with pragmas in the source code. This is the case of ImpulseC. The compilers considered use a variety of techniques for inserting clock cycle boundaries. Handel-C and SystemC use fixed implicit rules for inserting clocks and are very simple. Assignments and delay statements each takes one cycle in HandelC and instructions between two wait() statements take one cycle in SystemC. All the instructions inserted in a **par** statement are executed in one clock cycle in HandelC.

These simple rules can make it difficult to achieve a particular timing constraint. It is difficult to predict when a second input data can be inserted.

Either all FPGA elements are independent and the pipeline clock is one clock cycle or reuse is possible that makes the pipeline clock equivalent to the processing time. According to the data types, C-based Design tools considered several approaches. The first approach neither modify nor augment C's types but allow the compiler to adjust the width of the integer types outside the language. The second approach is to add hardware types to the C-language. Handel-C and ImpulseC compilers chose the data customization.

ImpulseC compiler allows automatic pipelining through pragmas but only for inner loops. Loop unrolling is used to obtain full pipelining. Precise control of the number of stages is difficult with such pragmas. Pipeline exploration is conducted automatically with VHDL synthesis on different solutions providing a frequency graph function of the latency/rate of the pipeline. This helps to obtain the higher rate/latency pipeline but with no considerations of the area. It is thus difficult to make a compromise between timing and area constraints. Also RAM/register inference selection is only obtained through a compilation option, that is for all the design and not separately for each array, which is really limiting as registers are a limited resource in FPGA.

VI. CONCLUSION

It clearly appears that heterogeneity call for high level abstract modeling while at the same time this very property requires taking into account precise implementation feedback. This puts into question the capacity of C-based tools to meet this challenge. In this paper, we have conducted a quantitative evaluation of the impact of C-based high level synthesis on general methodologies and framework. Results variations among the tools and their emphasis through synthesis options exploration challenge the modeling of C-based heterogeneous systems. We argue that implementation issues (area, frequency, floorplan) for large scale heterogeneous systems should be taken into account when using MoC modeling since currently the tools do not guarantee that high level concurrency semantics properties are preserved. Future work will extend the size of the case studies and automate the evaluation process.

Acknowledgement

This work is partially supported by the ISLE Cluster (EmSoC project) from the Region Rhone-Alpes, FRANCE.

References

[1] Edwards, S.; Lavagno, L.; Lee, E.A.; Sangiovanni-Vincentelli, A.; Design of embedded systems: formal models, validation, and synthesis, Proceedings of the IEEE, Vol,85, Issue 3, March 1997 Page(s):366 - 390

[2] Gupta, R.; Le Guernic, P.; Shukla, S.K.; Talpin, J.-P. (Eds.) , Formal Methods and Models for System Design A System Level Perspective 2004, X, 374 p., Hardcover ISBN: 978-1-4020-8051-7

[3] F. Herrera, E. Villar "A framework for heterogeneous specification and design of electronic embedded systems in SystemC", ACM Transactions on Design Automation of Electronic Systems, Vol,12, Issue 3 , August 2007.

[4] Eker, J.; Janneck, J.W.; Lee, E.A.; Jie Liu; Xiaojun Liu; Ludvig, J.; Neuendorffer, S.; Sachs, S.; Yuhong Xiong, "Taming heterogeneity - the Ptolemy approach", Proceedings of the IEEE Vol,91, Issue 1, Jan. 2003 Page(s):127 - 144

[5] G.Nicolescu, A.Jerraya, Global Specification and Validation of Embedded Systems Integrating Heterogeneous Components 2007, XII, 148 p., Hardcover ISBN: 978-1-4020-6151-6

[6] C. Brooks, E.A. Lee, X. Liu, S. Neuendorffer, Y. Zhao, H. and Zieng" Ptolemy II Heterogeoneous concurrent modeling and design in Java". Vol,1: Introduction to Ptolemy II". Tech. Memo. UCB/ERL M05/21. V5.0. July 2005. http://www.ptolemy.eecs.berkeley.edu./

[7] F.Balarin, Y. Watanabe,H. Hsieh,L. Lavagno, C. Passerone, A. and Sangiovanni-Vincentelli: "Metropolis: An integrated electronic system design environment",. IEEE Comput. Mag. April 2003.

[8] S. Huet, E. Casseau, O. Pasquier "Design Exploration HW/SW rapid prototyping for real-time system design". RSP, June 2005, Montreal, Canada.

[9] S. Abdelhalim, E. Bourennane and S. Aboukhechem, "Communication Interfaces Generation for Hw/Sw Architecture in the STARSoC Environment", IEEE International Conference , ReConFig'06

[10] I.S.Uzun, A.Amira "Design and implementation of an efficient reconfigurable architecture for image processing algorithms using Handel-C" 13th European Signal Processing Conference, 4-8th September 2005, Antalya, Turkey.

[11] ImpulseC Inc "Co-developper's user guide" www.impulseC.com, 2007

[12] Agility : http://www.celoxica.com/products/agility/default.asp

[13] Xilinx: www.xilinx.com