Static Estimation of Execution Times for Hardware Accelerators in System-on-Chips

M. Holzer and M. Rupp
Vienna University of Technology
Institute for Communications and RF Engineering
Gusshausstr. 25/389, 1040 Vienna, Austria

Abstract— **Early performance estimation of a System-on-Chip design is a key issue for a successful design methodology. One of the most important parameters is the run time of a function. Especially optimization techniques like hw/sw partitioning rely on those estimations. This paper presents a static analysis method in order to characterize a hardware acceleration unit regarding its run time. The performance of the presented method is shown on several examples from the embedded systems area and compared to results from high level synthesis.**

I. Introduction

The International Technology Roadmap for Semiconductors for 2001 (`http://www.itrs.net/`) has emphasized that early performance estimation is an essential step in any System-on-Chip (SoC) design methodology. Especially performance parameters like time, size, and power consumption can be as important as meeting the functional requirements [1]. Within the refinement process of an algorithm, starting from a high level description down to an implementation, different kinds of software properties, usually called metrics, can be identified (Figure 1). The accuracy of those metrics increases, the more hw details are known. Early design decisions have a drastic impact on the final system performance [2]. About 90% of the overall costs are determined in the first stages of a design, thus early design decisions have a much higher resulting cost span than design decisions taken at the end of the development time.

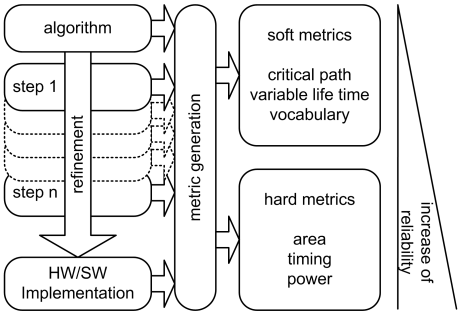

Fig. 1. Generation of hard and soft metrics.

As long as the hardware development is performed manually on the register transfer level (RTL), the required cycle count of the implementation is already known. With the introduction of high level synthesis tools [3] and thus the exploitation of potential parallelism, re-timing, and other techniques, prediction of the execution time has become more challenging.

Especially scheduling relies on estimations of timing in the area of real time systems in order to fulfill the required response time. Here also possible interference by other programs has to be taken into account, which results in a worst case response time (WCRT). For the task of hw/sw partitioning [4] execution time contributes to the cost function, which should be minimized. For this task, where heuristics could be used, already correct relations between two estimated values are sufficient in order to optimize a system.

Usually the timing analysis focuses on the prediction of the worst case execution time (WCET). However, the so called longest executable path [5] is not necessarily equal to the longest structural path. In this paper also the analysis of the best case execution time (BCET) and all other feasible paths for a function is presented in order to identify the run time interval of a process.

The rest of the paper is organized as follows. Section II gives an overview of related work in the field of execution time estimation. The design flow for developing complex SoC with the utilization of hardware accelerators (HA) is presented in Section III. This is followed by Section IV, where the static analysis of the timing estimation is described. In Section V the execution time estimation is performed on several examples and compared to the synthesis results of a high level synthesis tool. Finally, in the last section some conclusions are drawn.

II. Related Work

Execution time estimation can be achieved by simulation or static analysis. Simulation based approaches usually try to enrich the code with logging statements in order to obtain simulation traces. Hence, this procedure is appropriate for investigating the standard working conditions. Estimating the run time of a hardware implementation on an FPGA is achieved in [6] by simulating the algorithm and using a performance model of the FPGA. Also performance estimations for FPGAs are shown in [7]. In [8] reliable estimation of the execution time of an algorithm implemented in software running on a processor is presented. Analysis of the execution time achieved by synthesis of behavioral descriptions is given in [9].

This work has been funded by the Christian Doppler Laboratory for Design Methodology of Signal Processing Algorithms.

Static analysis allows for identifying the boundaries and usually gives a pessimistic estimate. Nevertheless, in the case of real time systems this is essential. Static analysis is usually applied in the software area. High level estimation without compiling for a target architecture requires the development of a virtual processor as shown in [10]. Usually such a processor model is not available and rather time consuming to develop. A commercial tool for WCET estimation is AbsInt (`http://www.absint.com/`) for software running on an ARM processor.

III. Hardware Accelerator Design Flow

The algorithm, which is described in a high level language (Matlab, Simulink, CoCentrics Sytem Studio), has to be mapped onto a hardware structure by performing hardware software partitioning. A general architecture of a SoC consists of one or more CPUs (DSPs and/or μPs) and HAs connected by a bus system. Those HAs are used to execute time critical tasks of the algorithm. They are already available as previously designed IP or it is necessary to develop them from scratch. Out of an algorithmic description usually a group of functions is selected to be implemented together inside one HA. For example, in Figure 2 the functions A, B, and C from the algorithmic description are grouped together to form the hardware accelerator *HA1* of the SoC. A bus interface has to be added as well to allow communication with the other units on the bus. The function E will be implemented in software running on the DSP. Either a manual or a tool supported partitioning process needs estimates of the final execution time properties of the selected functions inside the algorithmic description. Further refinement of the HA can be obtained by high level synthesis (e.g. Synopsys behavioral compiler, SPARK [11]).

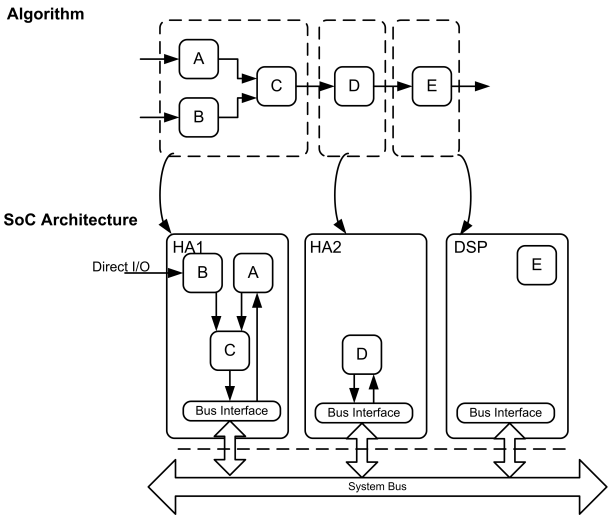

Fig. 2. Selection of functions for hardware accelerators in SoCs.

Especially in the case of integrating foreign IP into a design, timing characterization is an essential step in order to identify the usability in a project. An exchange method of those designs is for example described by the SPIRIT consortium (`http://www.spiritconsortium.com/`). Nevertheless, the full potential of capturing design parameters like timing can be only exploited, if those design properties are available to the designer and to all other tools in the design flow. For this reason it is necessary to store those properties persistently and to define interfaces, which allow for seamless access to the computed parameters. An integration of these design properties into a design database and its usage in a design flow is shown in the Open Tool Integration Environment (OTIE) [12], [13].

IV. Execution Time Estimation

Static analysis of a function is performed on its graph representation. A function can be decomposed into its control flow graph (CFG) built up with interconnected basic blocks (BB). Each BB contains a sequence of data operations ended by a control flow statement as last instruction (if, for, while). Those data operations can be represented as expression trees. A full timing characterization of the execution time of a hardware accelerator includes not only worst case estimation, but also best case estimation, as well as estimation of all other execution paths (Figure 3). This is accomplished by extracting all the possible paths from the CFG, starting at the root end ending at the exit node of the CFG. The computation of all possible paths seems to be feasible for functions restricted to a certain complexity, which is the case in a CFG derived from algorithms in industrial context.

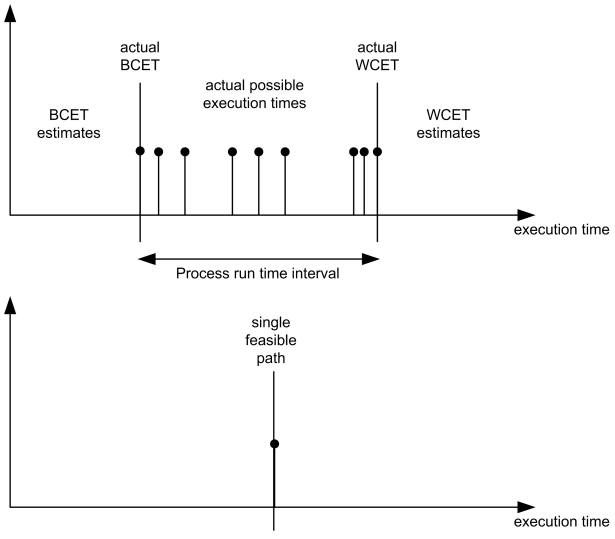

Fig. 3. Execution time of different execution paths of a function.

A process run time interval T_{int} can be identified by

$$T_{int} = WCET - BCET \tag{1}$$

Not all possible paths from the CFG contribute to the number of paths, but only those paths which are feasible. A path is feasible if the boolean product of its conditions is not false.

In Figure 4, a CFG is depicted with four paths:

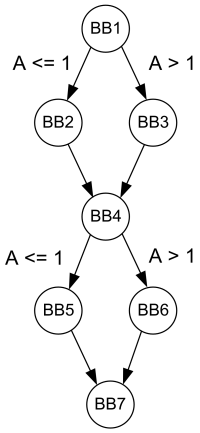

Fig. 4. Determining feasible paths of a CFG.

$$p_1 = (BB_1, BB_2, BB_4, BB_5, BB_7)$$
$$p_2 = (BB_1, BB_2, BB_4, BB_6, BB_7)$$
$$p_3 = (BB_1, BB_3, BB_4, BB_5, BB_7)$$
$$p_4 = (BB_1, BB_3, BB_4, BB_6, BB_7)$$

Only the paths p_1 and p_4 are feasible. Systems with a single feasible path (SFP) ($T_{int} = 0$) are hardware related algorithms e.g. FIR and FFT (Figure 3). In other words, the execution path is independent from the input data.

However, for context dependent paths, for each set of inputs, different paths can be examined. Systems with many feasible paths are control dominated, but differences between run times of different paths are experienced only if the spent execution time in the branches differs.

In order to estimate the cycle count that is needed to execute one path, the operations inside the BBs have to be considered. A complete sequential software solution would add all the cycle counts of the existing operations in the path. Hardware synthesis, especially if high level synthesis tools are used, exploits the possibilities of re-timing or loop invariant code motion. The target of detecting the fastest possible hardware implementation is to find the possible parallelism of each expression, by tree height reduction (THR) [14] (Figure 5). A lower bound on the depth of an expression tree can be given by $O(\log_2(N))$ where N denotes the number of nodes (operations) inside the expression tree.

V. Timing Estimation Example

The shown static analysis techniques are demonstrated on two examples. The *mpeg* algorithm has been chosen from the embedded systems library called MediaBench [15] and a part of the *cell searching* (CS) algorithm in UMTS from the mobile communication domain.

In order to compare the estimates to results, derived by high level synthesis, the SPARK [11] environment has been taken. For the synthesis, all optimization options have been turned on, like *loop invariant code motion*, which moves computations

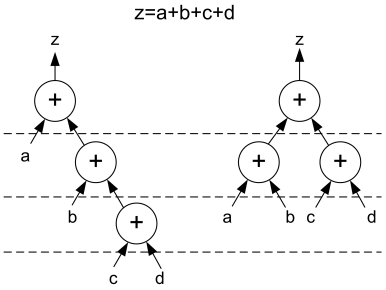

Fig. 5. Tree height reduction.

that are unchanged by the iteration of one or more surrounding loops out of the loops, thus eliminating redundant computation. Also *elimination of common sub expression* and *constant propagation* is deployed.

Function *CS*	$E_1(D)$	$E_2(D)$	$R_1(D)$	$R_2(D)$	A_1	A_2
Filter 1	228	230	165	167	0.62	0.61
Filter 2	2112	2114	1330	1333	0.41	0.41
Sqr and Sum	3	3	5	5	0.6	0.6
Slot Accu	2	2	1	1	0	0
Peak Detection	15360	15360	15362	15362	0.99	0.99
Function *mpeg*						
calcid	3	3	6	6	0.5	0.5
calculate fwd	13	13	15	17	0.76	0.86
predcase1	256	512	475	539	0.94	0.53
predcase2	2	960	4	1486	0.64	0.5

TABLE I

BCET AND WCET EXECUTION TIME PREDICTION.

In Table I the predicted cycle count for BCET E_1 and WCET E_2 for the functions of the chosen examples are shown. Also the high level synthesis results are given for the BCET R_1 and WCET R_2. The accuracy A (between estimated value and actual value) as defined in Equation 2 is also depicted.

$$A = 1 - \frac{|E(D) - R(D)|}{R(D)} \tag{2}$$

In regard of applying the cost estimation heuristic within transformational design space exploration, the ability to quantify relative dependencies of design characteristics is much more important than the ability to capture absolute values. For comparison of values it is only needed to achieve a high fidelity [16] value as proposed by Gajski.

$$Fidelity = 100\frac{2}{n(n-1)}\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\mu_{ij} \tag{3}$$

The fidelity measure supplies for a given set of n reference values $R_1, ..., R_n$ and n estimated values $E_1, ..., E_n$ a number describing the quality of the estimate with respect to its ability to quantify relative dependencies of pairs of reference/estimation values.

$$\mu_{ij} = \begin{cases} 1 & if \begin{cases} R_i < R_j \wedge E_i < E_j \\ R_i > R_j \wedge E_i > E_j \\ R_i = R_j \wedge E_i = E_j \end{cases} \\ 0 & otherwise \end{cases} \quad (4)$$

The fidelity value for the given set of nine function has been evaluated and gives for the WCET 0.96 and 1 for the BCET estimation values. Those values emphasize their usability for optimization processes, where only relative values are needed.

Further, Table II reports on the number of paths (N_{path}) found in the control flow graph and the reduced amount of paths (N_{fpath}) by selecting only the feasible ones. Also, the process run time intervals, estimated (E_{int}) and deduced by synthesis (R_{int}) are shown in this table.

Function *CS*	N_{path}	N_{fpath}	E_{int}	R_{int}
Filter 1	16	16	2	2
Filter 2	64	64	3	3
Sqr and Sum	1	1	0	0
Slot Accu	1	1	0	0
Peak Detection	2	2	0	0
Function *mpeg*				
calcid	1	1	0	0
calculate fwd	1	1	0	2
predcase1	16	4	256	64
predcase2	128	11	958	1482

TABLE II

NUMBER OF FEASIBLE PATHS AND PROCESS RUN TIME INTERVAL.

In Figure 6 the execution time profile of the function *predcase1* is depicted. In Table II is shown that 16 structural paths can be determined from the CFG of the function. Not all of them lead to different execution times. The dashed lines denote, those path which are not feasible and therefore do not contribute to the actual possible paths. So that after removing the non feasible paths only four paths remain. Only three paths are drawn solid, because two of the remaining paths lead to the same execution time of 320 cycles. The BCET has the value of 256 cycles and the WCET has 512 cycles with a run time interval T_{int1} of 256 cycles. Whereas for the non feasible paths a BCET of 128 cycles and a WCET of 896 cycles (T_{int2}=768 cycles) can be determined. The removal of the non feasible paths leads to much tighter WCET and BCET estimates, so that the estimated run time interval has been significantly reduced.

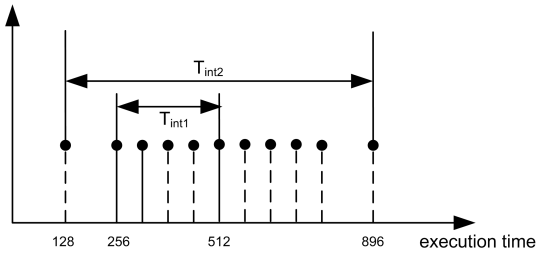

Fig. 6. Execution time profile for the predcase1 function from the *mpeg* algorithm.

VI. CONCLUSIONS

Early and reliable estimation of performance parameters is a necessary step for reducing time to market in an SoC design methodology. Especially design decisions for the design of a complex SoC like hw/sw partitioning need reliable information of the timing of HAs, as well as the scheduling of real time systems. The presented static analysis approach has been applied to several functions out of the embedded systems area. It has been shown that in cases of relative values, a high fidelity measure can be achieved. Also removing of the non feasible paths has demonstrated, that the estimates especially for WCET can be tightened, which allows for better utilization of the system resources.

REFERENCES

[1] A. Sangiovanni-Vicentelli and G. Martin, "Platform-Based Design and Software Design Methodology for Embedded Systems," in *IEEE Design & Test of Computers*, vol. 13, December 2001, pp. 23–33.

[2] Y. L. Moullec, P. Koch, J.-P. Diguet, and J. L. Philippe, "Design Trotter: Building and Selecting Architectures for Embedded Multimedia Applications," in *IEEE International Symposium on Consumer Electronics*, December 2003.

[3] D. Gajski, N. Dutt, A. Wu, and S. Lin, *High-Level Synthesis: introduction to chip and system design*. Kluwer Academic Publishers, 1992.

[4] B. Knerr, M. Holzer, and M. Rupp, "HW/SW Partitioning Using High Level Metrics," in *International Conference on Computing, Communications and Control Technologies (CCCT)*, August 2004, pp. 33–38.

[5] P. Altenbernd, "On the false path problem in hard real-time programs," in *Euromicro Workshop*, June 1996, pp. 102–107.

[6] P. Bjuréus, M. Millberg, and A. Jantsch, "FPGA resource and timing estimation from Matlab execution traces," in *Proceedings of the tenth international symposium on Hardware/software codesign*, 2002, pp. 31–36.

[7] R. Enzler, T. Jeger, D. Cottet, and G. Tröstler, "High-Level Area and Performance Estimation of Hardware Building Blocks on FPGAs," in *R.W. Hartenstein and H. Grünbacher (Eds.) FPL 2000*, vol. 1896. Springer, 2000, pp. 525–534.

[8] H. Posadas, F. Herrera, P. Sánchez, E. Villar, and F. Blasco, "System-Level Performance Analysis in SystemC," in *Design, Automation and Test in Europe*, Feb 2004, pp. 378–384.

[9] D. Mintz and C. Dangelo, "Timing Estimation for Behavioral Descriptions," in *International Symposium on System Synthesis*, 1994, pp. 42–47.

[10] P. Giusto, G. Martin, and E. Harcourt, "Reliable estimation of execution time of embedded software," in *Proceedings of the conference on Design, automation and test in Europe*, 2001, pp. 580–589.

[11] S. Gupta, "Spark: A high-level synthesis framework for applying parallelizing compiler transformations," in *International Conference on VLSI Design*, January 2003.

[12] P. Belanović, M. Holzer, D. Mičušík, and M. Rupp, "Design Methodology of Signal Processing Algorithms in Wireless Systems," in *International Conference on Computer, Communication and Control Technologies CCCT'03*, July 2003, pp. 288–291.

[13] P. Belanović, B. Knerr, M. Holzer, G. Sauzon, and M. Rupp, "A Consistent Design Methodology for Wireless Embedded Systems," in *accepted for publication EURASIP Journal of Applied Signal Processing*, 2005.

[14] D. Kuck, *The Structure of Computers and Computation*. John Wiley & Sons, 1978.

[15] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "MediaBench: a tool for evaluating and synthesizing multimedia and communication systems," in *International Symposium on Microarchitecture*, 1997, pp. 330–335.

[16] D. Gajski, F. Vahid, S. Narayan, and J. Gong, *Specification and Design of Embedded Systems*. Prentice Hall, Englewood Cliffs, 1994.