Static Code Analysis of Functional Descriptions in SystemC

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

Abstract

The co-design of hardware and software systems with object oriented design languages like SystemC has become very popular. Static analysis of those descriptions allows to conduct the design process with metrics regarding quality of the code as well as with estimations of the properties of the final design. This paper shows the utilization of software metrics and the computation of high level metrics for SystemC, whose generation is embedded into a complete design methodology. The performance of this analysis process is demonstrated on a UMTS cell searching unit.

1. Introduction

For the design of a signal processing system consisting of hardware (hw) and software (sw) many different programming languages have been introduced like VHDL, Verilog, and C/C++. One of the latest major contributions to the field of hw/sw co-design languages is SystemC (`www.systemc.org`) introduced by the Open SystemC Initiative (OSCI) in 1999. This language allows the designer to describe a system on different levels of abstraction. Those start with an un-timed description of the algorithm, which is ideally refined until the final implementation has been reached, while still staying in the same development environment. During this refinement process, it is of paramount importance to base design decisions on reliable characteristics and to assure quality of the written code. Those characteristics of the code, usually called metrics, are early estimates of the properties of the final implementation. Those estimates must satisfy three criteria: accuracy, fidelity, simplicity [9]. Early design decisions have a huge impact on the final system performances [18], about 90% of the overall costs are determined in the first stages of a design. Figure 1 depicts the evolution of the cost during the development time [2], where it can be seen that early design decisions have a much higher resulting cost span than design decisions taken at the end of the development time.

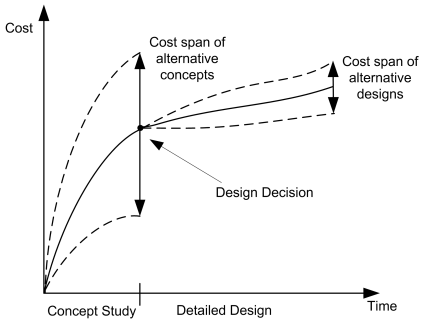

Figure 1. Evolution of the effort and cost during the development time.

Because of the lack of supporting tools many high level design decisions are taken on the basis of the experience of designers and not based on well defined metrics. The definition of those metrics can help to produce more reliable and traceable decisions. With the growing complexity of signal processing algorithms, especially in the wireless domain, also the need for automatic analysis of the algorithm arises. Especially tasks like partitioning of a system into hw and sw, analysis of bit widths, and mapping of descriptions to architectures deploy algorithms based on metrics. For solving those usually NP-hard problems, heuristics like genetic algorithms or tabu search are used, which heavily depend on cost functions, built up with metrics like execution time, area, and power consumption.
High level metrics can be generated by synthesis of the design, which gives accurate estimation, but is usually a time consuming task. Often only critical parts are modelled within a rapid prototyping environment in order to measure worst case scenarios, with the disadvantage that the target architecture has different properties than the prototyping target (e.g. timing). Another possibility is a simulation based approach of a design, which allows for esti-

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

mations. As the time spent on verification of a system increased already up to 80% of the design time [13] reduction of the simulation effort is one of the most eminent design targets. This paper focuses on static code analysis (SCA) techniques, which gives the possibility for fast generation of metrics.
The rest of the paper is organized as follows. Section 2 points out some existing contributions in the field of high-level metrics. Section 3 gives an overview of metrics that can be used for the investigation of SystemC and can be obtained by means of SCA. This is followed by Section 4 describing the integration of the analysis into a consistent design methodology. The Section following this covers an application example of a UMTS cell searching algorithm and reports on the achieved results. In the last section conclusions are made and perspectives for the future work are given.

2. Related Work

Within the refinement process of an algorithm, which passes through several intermediate steps, starting at a high level description and ends in a hw/sw implementation different kinds of metrics can be identified (Figure 2). The accuracy of those metrics increases the more hw details are known.

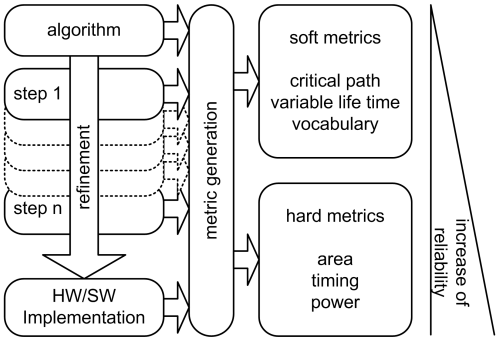

Figure 2. Generation of hard and soft metrics.

A first analysis can be performed rapidly without knowing any architectural details. The usage of these metrics already has quite a long history in the development of sw. Different metrics have been defined for functional programming languages like Pascal or C. One of the most popular metrics, known as *cyclomatic complexity*, has been defined by McCabe [16], expressing readability and testability of an implemented function. Another contribution to static analysis of code has been defined by Halstead [11], which focuses on predicting the design effort for a sw module. With the development of object oriented languages like C++ and Java, metrics for the investigation of object oriented features have been introduced, which can be reused for SystemC [5]. Nevertheless those metrics need to be reinterpreted in the context of hw/sw co-design.
Whereas sw metrics focus on design quality, testability, and design effort, the usage of hw related metrics tries to identify estimates of the final implementation like hw size (area) [4], power consumption [7], or execution time [20]. First attempts for investigating hw description languages have been achieved for VHDL. Approaches for identifying slices, which are functional dependent code parts and its application to hw description languages, especially VHDL, is given by Clarke et al. [6]. The derivation of function points for VHDL is presented by Fornaciari et al. [8]. Brandolese et al. [3] demonstrates extraction of metrics from SystemC descriptions for FPGAs. Another contribution based on SystemC is given by Agosta et al. [1], where a formal method for the analysis of transaction level models is derived, with focus on communication effort, memory size, and synchronization metrics. Gerlach and Rosenstiel [10] present high level design transformations based on control data flow graphs (CDFG) for the optimizations of C code for hw/sw co-design.

3. Metrics

The description of an algorithm in an object oriented language like SystemC allows the definition of two different kinds of metrics structure related metrics, which describe the hierarchical properties of the description and function related metrics, which characterize the implementation of the algorithm.

3.1. Structure Related Metrics

Structure related metrics describe the hierarchical properties of the SystemC description. Many of those metrics are defined for object oriented languages and can be reused for SystemC. Most prominent candidates are the number of classes, the number of instances, the maximum depth of hierarchy, and the number of methods.

The depth of hierarchy is a well known concept for hw description languages like VHDL. For maintenance reasons and understandability, the maximum depth is usually restricted to three hierarchy layers.

3.2. Function Related Metrics

Function related metrics refer to the characteristics of the implementation of the algorithm in a specific language. Most of these characteristics can be investigated on graph structures that represent the code. In general, the algorithm inside a SystemC module, written in form of sequential code, can be decomposed into its control flow graph (CFG),

built up of interconnected basic blocks (BB). Each BB contains a sequence of data operations ended by a control flow statement as last instruction. This sequence of data operations itself forms a data flow graph (DFG) or equivalently one or more expression trees. A control flow graph is a directed graph with only one *root* and one *exit*. A *root* defines a vertex with no incoming edge and the *exit* a vertex with no outgoing edge. Due to programming constructs like loops those graphs are not cycle-free. Figure 3 shows an example of a function and its graphical descriptions. The concentric circle vertex denotes the start of the function, while the pentagonal one depicts its end.

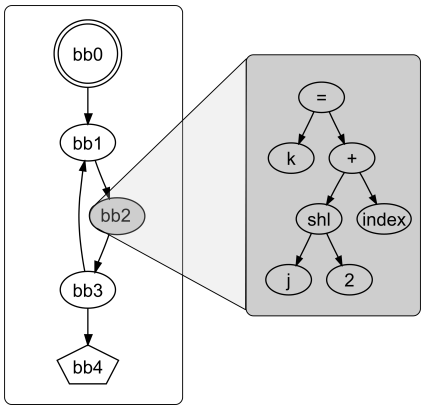

Figure 3. Control flow graph (CFG) and expression tree of one basic block.

3.3. Cyclomatic Complexity

McCabe [16] defined the metric *cyclomatic complexity* V. It can be used for determining the effort for structural testing of a function, and therefore as a metric for complexity. The *cyclomatic complexity* value expresses the number of linearly independent paths. A new path is independent from other paths if it is a path through the program that introduces at least one new processing statement or a new condition. Hence, if a path coverage of 100% should be achieved at least V test cases have to be performed. *Cyclomatic complexity* of a graph G can be computed as shown in Equation 1.

$$V(G) = e - n + 2 \tag{1}$$

In this equation e represents the number of edges and n the number of nodes of the graph G. Values of V less than 10 are considered as functions of minor complexity, with low verification effort, whereas higher *cyclomatic complexity* is considered as error prone and, in particular in the case of hw implementation where a testing coverage of 100% is aimed, testing effort increases dramatically.

3.4. Degree of Parallelism

The degree of parallelism γ describes the relation between the number of operations N_{op} (arithmetic, logic and relational as defined in Table 1) versus the number operations N_{opl} found in the longest path of the algorithm's CFG.

Operation type	Operation
arithmetic operation	ADD, SUB, MUL, DIV
logic operation	OR, XOR, AND
relational operation	$=, \neq, <, >, \leq, \geq$

Table 1. Different operation types.

$$\gamma = \frac{N_{op}}{N_{opl}} \tag{2}$$

Functions with a low γ value are rather sequential, whereas function with a large γ value offer in the case of non pipelined implementations possibilities for resources reuse. Nevertheless, in order to explore the reuse capabilities for each kind of operation, a longest path for each type of operation has to be identified. All basic blocks can be annotated with a set of different weights $W(BB_i) = \{w_1, w_2, .., w_m\}$, $i = 1..N$ that describe all its internal operations (e.g. w_1 = number of ADD operations). The longest path dependent on the jth distinct weight, LP_j is that sequence $(BB_k\, BB_l\, ..\, BB_m)^T$ of BBs through the CFG, which yields a maximum path weight PW_j by summing up all the weights w_j of the basic blocks that belong to this path. This is shown in Equation (3).

$$PW_j = \sum_{\substack{i=1 \\ BB_i \in LP_j}}^{N} w_j(BB_i) \tag{3}$$

With this PW_j a γ_j reflecting the reuse capabilities of each kind of operation j can be defined.

$$\gamma_j = \frac{N_{op_j}}{PW_j} \tag{4}$$

3.5. Memory and Control Metrics

Analysis of the memory behaviour has been specified in [17, 18] with the Memory Orientation Metric (MOM).

$$MOM = \frac{N_{mac}}{N_{op} + N_{cop} + N_{mac}} \tag{5}$$

Here N_{op} defines the overall number of operations as defined in Table 1, N_{cop} the number of control statements (if, for, while), and N_{mac} the number of memory accesses. A MOM value near one identifies a function with high memory usage. In order to use this metric for power consumption, where different energy models are used for read and

write operations [12], a further differentiation results from distinguishing between read and write accesses.

$$MROM = \frac{N_{mrac}}{N_{op} + N_{cop} + N_{mac}} \quad (6)$$

$$MWOM = \frac{N_{mwac}}{N_{op} + N_{cop} + N_{mac}} \quad (7)$$

In Equation 6 N_{mrac} represents the number of memory read operations and in Equation 7 N_{mwac} the number of memory write operations.
In contrast to the memory metrics, a control orientation metrics (COM) identifies whether a function is dominated by control operations.

$$COM = \frac{N_{cop}}{N_{op} + N_{cop} + N_{mac}} \quad (8)$$

When the COM value tends to be 1 the function is dominated by control operations. This is usually an indicator that an implementation of a control-oriented algorithm is more suited for running on a controller than to be implemented as dedicated hw.

4. Analysis Environment

In order to provide the results of the SCA in an open environment to other tools, the analysis process is embedded into a unified design environment, which allows the use of various EDA tools as they are being favoured by the various design teams. A so-called single system description (SSD) has been established in form of a design database [14]. Figure 4 depicts the SSD encircled by a selection of EDA tools and self-developed optimization libraries. Dedicated system description interfaces (SDI) facilitate the seamless communication between the heterogeneous system components and the SSD.

The SCA is performed after reading in the algorithmic description written in SystemC over an SDI into the SSD. After that, in the first step, static code analysis by the Open Compiler Environment (OCE) from ATAIR (`www.atair-software.com`) is performed. This sw decomposes each SystemC function of the design into its basic blocks and allows for the construction of the CFG and DFG representation. Beside its tables for storing the structure of the design, the SSD also provides tables for storing basic blocks and operations for every process. In the second step, the generation of the metrics is achieved by means of a C++ graph class. The results of the SCA are persistently saved in property tables. Further refinement steps can take place within this environment based on the stored metrics information. Those steps include tools for automatic partitioning of the system into hw/sw parts [15], automatic generation of virtual prototypes [14], and an environment called *fixify* for

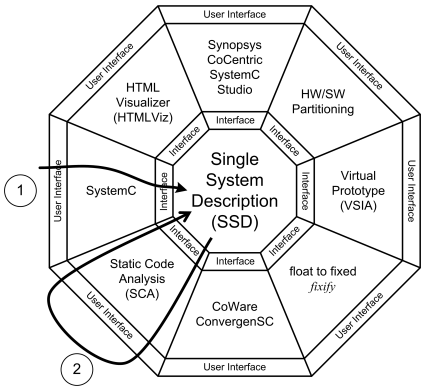

Figure 4. Importing of SystemC into the Single System Description and metrics generation.

performing the task of fixed-point to floating-point conversion [19].

The stored metrics allow for the construction of a cost function. In general a cost function C for a hw realization of a function can be constructed by the sum of metrics x_i with the weight w_i.

$$C = \sum_{x_i \in X} w_i x_i \quad (9)$$

Usually the metrics x_i represent normalized values for timing, area, and power. Additionally those metrics have constraints given by functional and economical requirements. Within this cost function also cyclomatic complexity V can be taken into account, in order to consider the cost for verification effort.

A further application of the presented metrics is its usage for the hw/sw partitioning process. Here a huge search space demands for heuristics that allow for partitioning with a reasonable time effort. Nevertheless, a reduction of the search space can be achieved by assigning certain functions to hardware or software beforehand. This can be accomplished by an affinity value A as shown in Equation 10.

$$A = COM^{-1} + \sum \gamma_j \quad (10)$$

A high value A and thus a high affinity to a hardware implementation is caused by less control operations and high parallelism of the the used kind of operations.

5. Example

An algorithmic description written in SystemC of a UMTS cell searcher algorithm has been processed by the SCA framework in order to gather the presented metrics. This cell searcher performs the slot synchronization part of

the cell searching procedure, by detecting the start of a slot transmitted by the base station in a UMTS network.

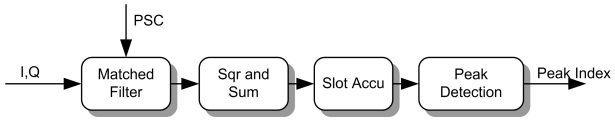

Figure 5. Block diagram of the cell searcher.

The cell searcher implementation consists of four SystemC modules as depicted in Figure 5. The *I* and *Q* values are correlated within the *Matched Filter* with the chip sequence of the primary synchronization code (PSC). Two different types of matched filter functions *Matched Filter 1* and *Matched Filter 2* have been implemented in order to show design tradeoffs by means of SCA. The functions *Square and Sum* together with *Slot Accu* perform addition and accumulation over several slots of the energy values. As last step, the *Peak Detection* sorts the energy values and searches for the highest peak, whose index determines the start of a slot.

After importing of the cell searcher project into the SSD, SCA has been performed. This example consists of four classes, each instantiated once. It has a flat hierarchy, so that the maximum depth of hierarchy is only one. Each class contains one method. The results of the SCA of the code are shown in Table 2.

Function	MOM	MROM	MWOM	COM	V	γ
Sqr and Sum	0.62	0.5	0.12	0	1	1
Slot Accu	0.67	0.45	0.22	0	1	1
Peak Detection	0.39	0.26	0.13	0.11	5	1
Matched Filter 1	0.49	0.35	0.14	0.16	7	1.25
Matched Filter 2	0.55	0.35	0.2	0.16	9	1.29

Table 2. Metrics for control, memory usage, cyclomatic complexity, and parallelism.

The functions *Sqr and Sum* and *Slot Accu* show similar memory usage and contain no control parts. *Peak Detection* exhibits less memory accesses than the functions *Sqr and Sum* and *Slot Accu*, but has a minor fraction of control statements. Both matched filter implementations show the same effort for control and read operations, but the *Matched Filter 2* has more write accesses compared to the first implementation.

The functions *Sqr and Sum* and *Slot Accu* are rather simple functions with less operations and operands, as well as no control and no parallelism. *Peak Detection* performs a shell sort algorithm, which shows the need for control operations, but also gives no possibilities for resource sharing. A comparison of the two matched filter implementations regarding their vocabulary indicates, like the memory orientation metrics a higher memory usage by a higher number of distinct operands in the *Matched Filter 2* design. The reuse capabilities of the two matched filter implementations regarding the utilized types of operations (ADD, SUB) tend to be one ($\gamma_{ADD} = 1.24$, $\gamma_{SUB} = 1$), because both types of operations appear in the longest paths much more frequently than in the other possible paths.

Figure 6 depicts the CFG of the matched filter implementations *Matched Filter 1* and *Matched Filter 2*.

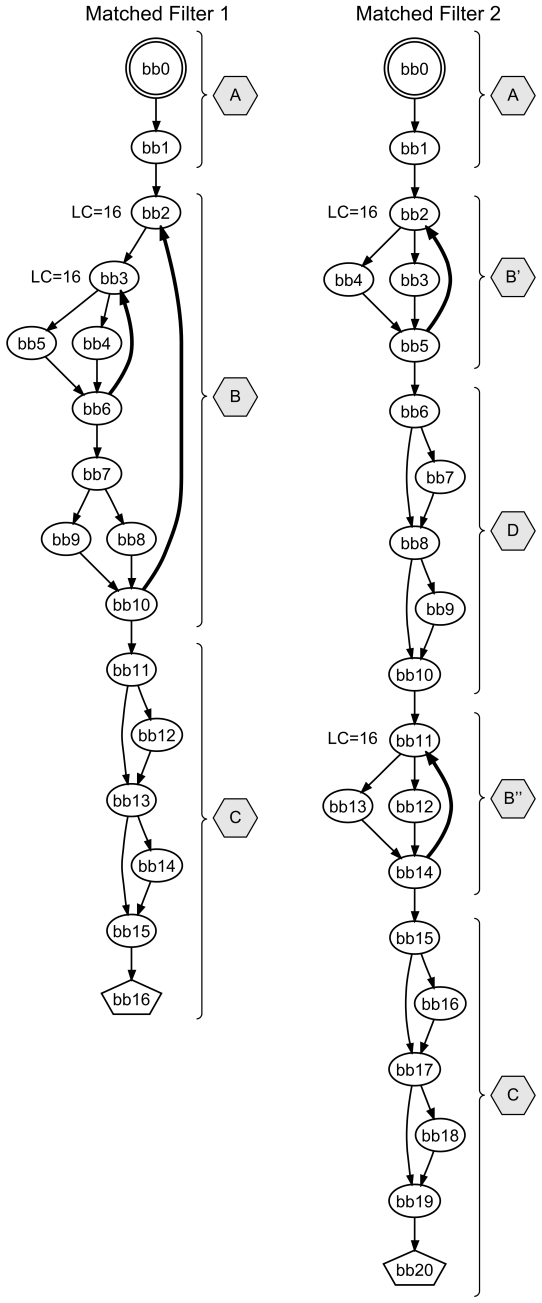

Figure 6. Control flow graphs of two different matched filter implementations.

Both implementation share the CFG regions A and C. *Matched Filter 1* consists of two nested loops in region B, both with a loop count (LC) of 16, which forces the innermost basic blocks (BB4, BB5) to be executed 256 times. Those loops are decoupled in the second implementation within the region B' and B". This is achieved by storing intermediate results, which causes an additional memory effort of storing 512 values, but necessitates much less computation in its longest computation path. Also, additional control effort is introduced as denoted within region D. This causes a higher *cyclomatic complexity* value and therefore slightly increases the testing effort for this implementation. While both implementations are regarded as low complexity functions, it can be seen that a tradeoff between testing effort and implementation cost exists. The metrics have been used to derive hw and sw execution times, as well as area consumption in order to construct a cost function, which can be used for hw/sw partitioning based on heuristics. The results of this partitioning are shown in [15].

6. Conclusions

The need for metrics for high level design decisions in the design of embedded systems has emerged, in particular in the field of system on chip for wireless communication systems. Static code analysis of SystemC provides a possibility for fast generation of structural information for the design, in order to make appropriate design decisions. This paper shows the application of sw metrics for the usage in hw/sw co-design. The derivation of these metrics is integrated in an environment as part of a whole design flow and allows the usage of those metrics by other optimizations tools. An example is presented with its metrics generation. Furthermore, design trade-offs are shown on different implementations of a matched filter design.

References

[1] G. Agosta, F. Bruschi, and D. Sciuto. Static Analysis of Transaction-Level Models. In *Design Automation Conference*, pages 448–453, June 2003.

[2] J. Axelsson. Cost Model for Electronic Architectures Trade Studies. In *Sixth International Conference on Engineering of Complex Computer Systems*, 2000.

[3] C. Brandolese, W. Fornaciari, and F. Salice. An Area Estimation Methodology for FPGA Based Designs at SystemC-Level. In *Design Automation Conference*, pages 129–132, June 2004.

[4] K. Büyüksahin and F. Najm. High-Level Area Estimation. In *International Symposium on Power Design and Electronics (ISLPED'02)*, pages 271–274, August 2002.

[5] S. R. Chidamber and C. Kemerer. A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering*, 20:476–493, 1994.

[6] E. M. Clarke, M. Fujita, S. P. Rajan, T. Reps, S. Shankar, and T. Teitelbaum. Program Slicing of Hardware Description Languages. In *Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, pages 298–313, September 1999.

[7] W. Fornaciari, P. Gubian, D. Sciuto, and C. Silvano. Power Estimation of Embedded Systems: A Hardware/Software Codesign Approach. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 6:266–275, 1998.

[8] W. Fornaciari, F. Salice, U. Bondi, and E. Magnini. Development Cost and Size Estimation Starting from High-Level Specifications. In *9th International Workshop on Hardware/Software Co-Design (CODES 01)*, pages 86–91, 2001.

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

[10] J. Gerlach and W. Rosenstiel. Development of a High-Level Design Space Exploration Methodology. Technical report, Tübingen, Wilhelm-Schickard-Institut für Informatik, 1999.

[11] M. H. Halstead. *Elements of Software Science*, volume 7. Elsevier, 1977.

[12] M. B. Kamble and K. Ghose. Analytical Energy Dissipation Models For Low Power Caches. In *International Symposium on Low Power Electronics and Design*, pages 143–148, 1997.

[13] M. Keating and P. Bricaud. *Reuse Methodology Manual for System-on-Chip Designs*. Kluwer Academic Publishers, 1998.

[14] B. Knerr, M. Holzer, P. Belanović, G. Sauzon, and M. Rupp. Advanced UMTS Receiver Chip Design Using Virtual Prototyping. In *International Symposium on Signals, Systems, and Electronics (ISSSE)*, August 2004.

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

[16] McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2:308–320, 1976.

[17] Y. L. Moullec, N. B. Amor, J.-P. Diguet, M. Abid, and J.-L. Philippe. Multi-Granularity Metrics for the Era of Strongly Personalized SOCs. In *Design, Automation and Test in Europe*, pages 674–679, March 2003.

[18] 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.

[19] P. Belanović and M. Rupp. Automated Floating-point to Fixed-point Conversion with the *fixify* Environment. In *International Workshop on Rapid System Prototyping RSP'05*, June 2005.

[20] Per Bjuréus, M. Millberg, and A. Jantsch. FPGA Resource and Timing Estimation from Matlab Execution Traces. In *International Workshop on Hardware/Software Co-Design*, pages 31–36, May 2002.