The 19th IEEE/IFIP International Symposium on Rapid System Prototyping

High-Level Estimation of Execution Time and Energy Consumption for Fast Homogeneous MPSoCs Prototyping

Sérgio J. Filho, Alexandra Aguiar, César A. Marcon, Fabiano P. Hessel
Av. Ipiranga, 6681 - PPGCC / FACIN / PUCRS, Porto Alegre, Brazil
{sergiojf, alexandra.aguiar}@inf.pucrs.br, {cesar.marcon, fabiano.hessel}@pucrs.br

Abstract

In order to fulfill the increasing performance requirements, complex embedded systems design makes use of many processors communicating through efficient infrastructures, performing Multiprocessor-Systems-on-Chip (MPSoCs). Issues related to execution time and energy consumption estimations become more relevant during the design stage of such systems, in order to verify their compliance with the specification. Different estimation techniques have been proposed, including analytical and simulation-based methods. Analytical methods are faster than simulation-based methods, but the system description is more complex, and sometimes this approach conducts to low precision results misleading future design steps. On the other hand, the more accurate results achieved with simulation-based method, using low-level descriptions, may delay the design making it unfeasible or at least affecting the time-to-market. In this context, improvements in simulation-based methods become pertinent. This paper presents a study, a design flow and a tool for high-level simulation-based estimation of execution time and energy consumption of homogeneous MPSoCs. The implemented tool, which employs the methodology presented in this paper, improved dramatically simulation times when compared to RTL simulations. The preliminary results show that, for some cases, the RTL simulation takes tens hours while the implemented tool gets close estimation results in just few seconds.

1 Introduction

Advances in fabrication technology allow the implementation of complex systems with millions of transistors. Such advances also allow the integration of multiple components, such as processors, controllers and memories integrated into a single chip, performing a System-on-Chip (SoC).

The design flow for embedded systems must consider real time execution, performance and power consumption requirements. Moreover, the market pressure requires fast, optimized and low cost designs. Flexibility is another important requirement. It must be possible to aggregate new functions to the SoC without re-designing it.

Microprocessors have an important role in the flexibility of embedded systems. SoC solutions use one or more processors from the same or from different families. Such solutions, named Multiprocessor-Systems-on-Chip (MPSoCs), require specific models and tools to deal with the complexity of current applications.

In order to evaluate the requirements of an MPSoC, hardware and software components functionalities have to be evaluated concomitantly. Nowadays, many of the available frameworks allow integrating these components later in the design flow - when the hardware prototype is available. However, since designers need to evaluate different implementation scenarios as soon as possible, estimation tools must be incorporated into the initial steps of the design flow.

These tools can be classified as simulation-based or analytical-based according to the underlying method. Analytical methods allow foreseeing a future state of the system, and sometimes its characteristics, by its partial or total behavior modeling facing of a set of input events. In this class of tools, the system does not need to pass by intermediate states. On the other hand, to simulation-based tools estimate a future state, it is necessary to pass through many intermediary states.

Normally, analytical-based tools are many times faster than simulation-based ones, making them attractive to complex systems estimations. However, those tools normally pay the cost of low accuracy when compared to simulation methods.

The objective of this work is to maintain the accuracy achieved by simulation-based tools, which uses low-level descriptions, together with the performance in execution time of analytical-based tools. In this sense, this work proposes a platform, a design flow and a tool for execution time and energy consumption estimation of homogeneous MPSoCs. A platform, composed of four processors interconnected by a bus system and described in VHDL is used as a study case for estimation tool evaluation. After synthesizing the VHDL high-level description of the platform into a gate-level (RTL) description, time execution

978-0-7695-3180-9/08 $25.00 © 2008 IEEE
DOI 10.1109/RSP.2008.25

and energy consumption estimations are taken from RTL descriptions and organized in a high-level tool, to speedup simulation times.

This work is organized as follows. Section 2 presents the state-of-art of MPSoC architectures and tools. Section 3 presents an MPSoC estimation platform, which is used as case study of our estimation tool. Section 4 describes our estimation tool. Section 5 presents preliminary results and in Section 6, conclusions are commented.

2 State-of-Art

Estimations achieved by simulation-based methods may be done with different levels of description. Some simulation tools use architectural description languages, such as LISA [1], Expression [2] and MIMOLA [3], to describe the processor architecture. Tools, which support such languages, produce the simulator, the compiler and sometimes the synthesizable hardware, according to the architecture description. Some commercial tools, such as Lisatek [4] and Maxcore [5] use LISA language.

Energy aware architectural design exploration and analysis simulation tool for ARM based SoC designs is proposed in [6]. The tool integrates the behavior and energy models of several user-defined custom processing units, as an extension to the cycle-accurate instruction-level simulator for the ARM low-power processor family. Although several studies demonstrate that technology, layout and gate level techniques offer energy consumption savings of a factor of two or less, architecture optimizations can often result in very expressive energy consumption savings [7]. In this sense, our work relates how a higher abstraction level tool can be used to model different application scenarios.

Analytical methods are useful for the solution space exploration in higher abstraction levels (for example, transaction level). Usually, the application analysis extracts the number of different types of instructions [8]. After that, such instructions are mapped to a performance model, which is used to calculate their execution time. However, this kind of mapping, is not detailed enough to represent complex systems, as in the case of distributed applications, which have MPSoC as a sound candidate for target architecture.

In [9], the authors use a non-linear method for execution time estimation. With a determined set of benchmarks, a classifier extracts a functional signature vector for a virtual processor. This vector contains the types of instructions and the number of executions of each one.

A characterization-based macromodeling technique is presented in [10]. This one enables the extraction of fast high-level models of reusable software components, which is based on pre-characterization of more detailed models (e.g. cycle models or instruction models), which consume more computation time. The effort directed toward the construction of macromodels for a software module is paid off due to the large number of macromodel applications, when the module is simulated in the context of all the programs that include it. However, for such cases, new software modules do not have higher abstraction of software implementation. Therefore, the impact in simulation time may be too high.

In [11], the authors evaluate the annotation method using a set of virtual instructions to represent the object-code level instructions. This method transforms the C language code into a virtual instruction set. Each instruction has a cost associated to the target architecture, which is obtained through either simulation or statistical methods [8]. The method, however, does not consider all implications of a distributed system.

An important factor in real time performance estimation is to find critical execution paths, which can be achieved by worst-case execution time (WCET) techniques. In [12], the authors propose an analysis of a static method, using a technique named *implicit path enumeration*, stipulating the number of executions in each basic block, regarding to the WCET. The limits are calculated through linear equations obtained from structural application and architectural analysis. Nevertheless, this kind of technique does not take into account neither detailed execution time nor energy estimations.

In [13], the authors present a method used to reduce the linear equations trying to extract a unique possible path. Their method does not use WCET analysis, but intervals calculated taking into account a variable execution cost of a basic block. Conversely, the method, does not consider cycle-accurate estimations, which are quite relevant in complex systems, as software portions performance is dependent of hardware implementation.

This work employs simulation of high-level abstraction levels for software execution time and energy consumption estimations, which performs a simulation tool. A detailed model of the architecture, implemented in RTL, is used as estimation reference to a cycle-accurate simulator.

3 Estimation Platform Proposal

3.1 Architecture

Some decisions regarding to the adopted architecture had to be taken to create a platform for execution time and energy consumption estimations. Such decisions consider the microprocessors and interconnection structure adopted in the architecture, which are necessary for the construction of a prototype.

The MIPS I processor was adopted in this work due to some advantages: (i) available open source description, (ii) the easiness of multiple cores integration, and (iii) existence of high performance compilers for a large portion of embedded systems applications. Moreover, the MIPS architecture is generic enough to represent a great segment of embedded processors characteristics (i.e. RISC architectures). Thus, the same framework analyzed may be reused and extended for other processor families.

SoCs composed of few cores may be built around a bus interconnection easily, with low cost and without significant performance losses. As a result, custom bus architecture was used to build the study case platform proposed in this work, since this platform is composed of four cores. Although this was the target MPSoC architecture, it can be easily expanded or reduced, according to the application as the interconnection system is highly parameterizable. Such architecture has been successfully prototyped in hardware (using FPGAs).

3.2 Methodology

The methodology adopted in this work starts with a simulation tool to detect the logic modules activity. This is performed by using VHDL gate-level simulation. During this step, a module is stimulated by changing the logical values on its input ports. This module reacts according to these logic values, generating VHDL events on each internal logic gate of the module. Such events can be used to estimate the energy consumed on each gate.

The first step of this method is synthesizing the architecture behavioral VHDL description using a target technology library. In this work, the architecture was synthesized through Leonardo Spectrum tool [14] using *CMOS TSMC 0.35 μm technology*[1], producing a VHDL netlist. After synthesis, this netlist is converted by CAFES [15] tool, creating references to the energy estimation library. The CAFES tool takes into account the energy consumed on each gate, according to input events (transitions) and gate fanout.

The next step is the circuit simulation and it is performed by a VHDL simulator, such as ActiveHDL [16] and ModelSim [14]. Figure 1 shows the methodology flow used for energy consumption estimation in digital CMOS circuits. In this work, ten different logic gates were used. Therefore, the synthesis process is not restricted to generate only this set of gates.

[1] This technology is just an example. The library can be pre-characterized with any kind of technology.

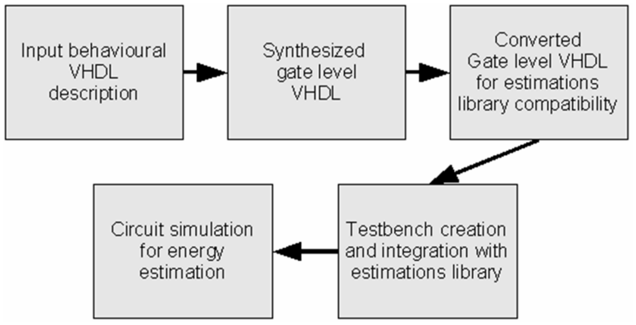
Input behavioural VHDL description
Synthesized gate level VHDL
Converted Gate level VHDL for estimations library compatibility
Testbench creation and integration with estimations library
Circuit simulation for energy estimation

Figure 1 – Methodology flow for execution time and energy consumption estimation.

The key component for energy consumption estimations is the pre-characterized library. The main underlying idea is the use of VHDL events to accumulate the power dissipated by the module under simulation. In fact, according to [15], the estimation does not report a real consumption, but a close value. The applied methodology also takes into account the power dissipated by parasite components, which are introduced by interconnections (metal and vias) and each logic gate fanout.

3.3 Single processor platform

Firstly, a platform with a single processor was built. Such platform consists of one Plasma [17] processor core, which implements the MIPS I instruction set and other components, as presented in Figure 2.

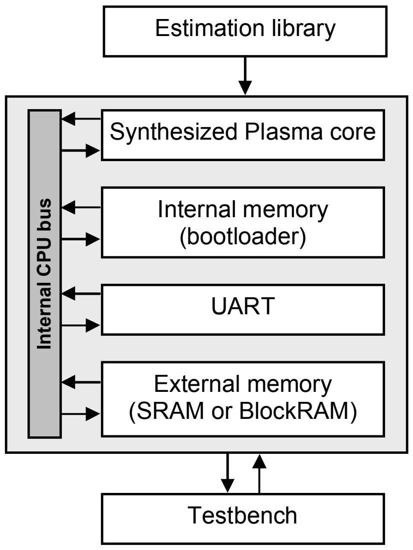

Figure 2 – Initial platform architecture.

Figure 2 shows the internal structure of the processor, formed by a netlist of the processing core, an internal memory, an UART and an interface to external memory.

The library is instantiated by the cores for energy consumption estimations.

The platform architecture conception isolates the processor core from the internal memory and UART. Thus, processor estimations refer only to its core (control, datapath and registers). The methodology flow was applied to this module enabling execution time and energy consumption estimations.

The application to be estimated is stored in the external core memory, which is generic, parameterizable, synthesizable and one cycle latency. Only the FPGAs BlockRAM resources limit its size.

All components communicate through an internal bus. A testbench generates clock and reset signals to the processor core and other modules. Therefore, the platform functionality is simulated and estimations can be obtained.

3.4 Complete platform

The complete platform used here as case study has four Plasma processors communicating through a media control access and a bus. Figure 3 presents the block diagram of the implemented architecture. Four processors and a bus compose this architecture, although the interfaces are defined in a very generic way and another communication structure, such as a network-on-a-chip (NoC) could be used instead. A more efficient communication structure was not used because the processors, in this configuration (without DMA), would not use its full bandwidth.

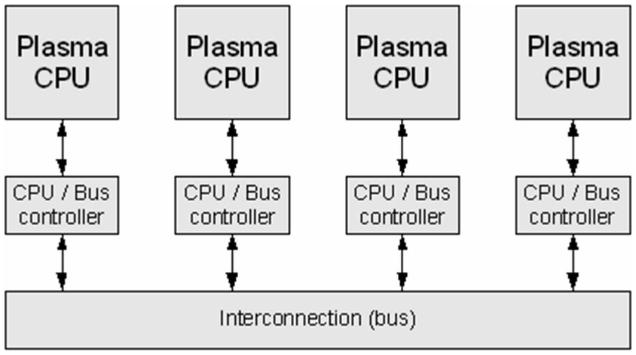

Figure 3 – Target architecture used for case study.

The platform was fully described in VHDL and prototyped in FPGA. Software drivers for communication were implemented, using simple *Send* and *Receive* blocking and non-blocking primitives.

The data exchanged among processors is organized in packets, whose structure is presented in Figure 4. The first 2 bits are used for signaling between processors and the control access. Another 6 bits are used for addressing and 24 bits hold data.

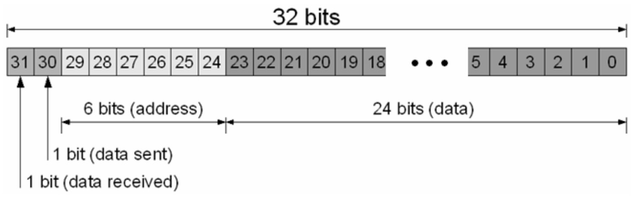

Figure 4 – Packet structure.

Our custom bus architecture was also described in VHDL, simulated and prototyped in FPGA. The bus architecture is economic in area and presents efficient throughput for the implemented MPSoC. Its characteristics are summarized on Table 1.

Table 1 – Bus characteristics

Feature	Description
Word width	8 bits (parameterizable)
Ports	4 (parameterizable)
Queue length	16 bytes (parameterizable)
Arbitration	Round Robin algorithm
Latency	2 cycles per word, after arbitration
Frequency	25 MHz (prototyped), max. 222 MHz
Size	8372 logic gates
Throughput	100 Mb/s (8 data bits @ 25 MHz)
I/O protocol	Handshake

The controller between cores and custom bus adapts the synchronous data port signals of each core to the asynchronous handshake protocol, implemented in the bus ports. This controller generates four 8-bit word bursts on the bus, for each 32-bit read or write event on the data ports of each core. Moreover, the controller also implements core addressing, which enables unicast or broadcasts packet destination address.

The proposed platform architecture isolates the processors from the interconnection structure. Such characteristic allows the measurement of each component separately after logic synthesis. Table 2 summarizes the logic used by each component of the implemented platform. Plasma processors represent 86.36% of the used logic and the remaining (13.64%) is used by the interconnection infrastructure.

Table 2 – Components area

Component	Size (logic gates)
Bus	8372
Processor / Bus controller	2893
Processor (Plasma core)	82348 (20587 each)
Total	**93613**

4 Estimation Tool

Gate-level simulations provide accurate execution time and energy consumption estimations [15]. However, the time spent to simulate even simple algorithms running on a single processor is prohibitive.

A sound estimation tool relies on the idea of achieving accurate results in affordable computation time. To achieve this goal, we developed a high-level simulation tool. An instruction set simulator (ISS) of the MIPS I architecture was implemented. Such simulator runs native object code, executing it according to the hardware implementation of Plasma processor. In addition, the simulator includes mechanisms for cycle counting and energy consumption estimation. The cycle counting mechanism is also responsible for the individual instruction accounting. Furthermore, a time simulation mechanisms were implemented, to emulate hardware execution over a finite amount of time. One of them is the number of cycles to be performed on real hardware execution.

Timing and functional emulation of the UART are also included. Consequently, algorithms that produce output to the UART are correctly emulated.

Different instructions generate different stimuli inside the processor core. Thus, the energy per cycle changes accordingly to these stimuli. The estimation tool computes this behavior, applying it during the simulation of the application. It reduces the estimation error of energy consumption.

Exhaustive analysis shows that the energy consumption estimation can be distributed according to different classes of instructions: (i) Arithmetic, (ii) Branches, (iii) Loads / Stores, (iv) Logical and (v) Moves / Shifts. This distribution is somewhat in high abstraction level, but efficient enough to achieve approximated energy consumption estimation. Further classes would improve energy consumption estimations, at the cost of higher simulation times. Data variance also influences in core energy consumption, consequently, the energy consumption per cycle was modified inside the ISS for exception cases.

Based on several RTL simulations, it was observed that different instructions inside a single class generated different energy parameters; as a result, weights were attributed for exception cases.

Energy parameters were obtained from RTL simulations (gate level), and included in the proposed tool, as detailed in the next section. Such parameters are part of the energy consumption mechanism.

5 Results

Application tests were created for each instruction class, and the energy parameters were obtained through VHDL simulations of the processor description. Each application includes several instructions of the same class. Such applications run for a limited time, repeating the instructions exhaustively. After this, the accumulated energy consumed by the processor is divided by the executed number of cycles, and the energy per cycle consumed by each instruction class can be obtained, as shows Table 3.

Table 3 – Average of energy consumption per cycle of each instruction class

Instruction class	Energy consumption per cycle (J)
Arithmetic	1.60864E-09
Branches	2.39897E-09
Load / Store	1.69180E-09
Logical	2.51948E-09
Move	1.92844E-09
Shift	2.92796E-09

Table 4 presents the estimated energy consumption of the interconnection structure, regarding to both intense packet traffic and idle situations. Application tests were created to simulate traffic generation and the interconnection structure was simulated in VHDL, likewise the processors were. The same drivers using simple blocking and non-blocking *Send* and *Receive* primitives were used on VHDL simulation and in our tool. The latency of packet traffic was also simulated in VHDL and included in our high-level estimation tool.

Table 4 – Interconnection energy consumption

Interconnection	Energy consumption (J)	
	per cycle	per packet
Idle	4.68900E-11	2.34450E-09
Data	1.62562E-10	8.12808E-09

Table 4 shows that packet traffic generates higher energy consumption of the interconnection in contrast to idle situations. Random data was generated in a test application and an average energy per packet was used as energy consumption reference.

Several applications were implemented or ported to the MIPS I architecture. A summary of applications simulated in the VHDL platform and on the estimation tool is presented on Tables 5 and 6.

Table 5 presents the execution time and the energy consumption estimations for applications running on a single processor.

Table 5 – Single processor benchmarks

Benchmark	VHDL		Our Tool		WCET Error	Energy Error
	WCET (ms)	Energy (J)	WCET (ms)	Energy (ms)		
Sobel	33.74	1.33E-03	34.11	1.62E-03	1.09%	17.87%
JPEG	205.33	11.10E-03	207.30	11.12E-03	0.95%	0.25%
CRC32	7.87	0.79E-03	7.87	0.77E-03	0.02%	2.65%
ISqrt	16.64	1.89E-03	16.640	1.96E-03	0.00%	3.37%
ADPCM	267.14	31.46E-03	267.14	25.75E-03	0.00%	18.15%
Random	83.14	7.82E-03	83.13	7.74E-03	0.00%	0.01%

Several benchmarks were used as case study. Time execution estimation is quite accurate, as shown on Table 5. The maximum error in this set of benchmarks was around 1%. Energy consumption estimations error was higher, around 18% on two corner cases, but other four cases demonstrated an estimation error of energy consumption below 4%.

The set of benchmarks is composed of several algorithms, being one of them an edge detection algorithm (Sobel) performed on a sample 32x32 pixel monochrome image, a baseline JPEG encoder used to encode a 32x32 pixel RGB sample, a standard CRC32 implementation, an integer square root algorithm, the Intel ADPCM encoder / decoder and a random number generator.

For a partial validation of the estimation tool, a non-trivial algorithm was chosen. As a case study, the JPEG encoder, conforming to ISO-10918, was implemented and ported to the MIPS I architecture. This encoder was implemented according to the reference standard [19]. The bitstream generated from the encoding process has been verified in conformance. A version of this algorithm, where the generated encoded bitstream was outputted to the UART, was tested and verified on the FPGA prototype.

Table 6 presents a benchmark of distributed applications, where algorithm parallelism and communication between processors occurs. The benchmark is composed of three algorithms – a broadcast algorithm, a distributed ADPCM encoder / decoder (MP ADPCM) and a distributed JPEG encoder (MP JPEG).

Table 6 – Benchmark for execution time and energy consumption estimation of distributed applications

Benchmark	VHDL		Our Tool		WCET Error	Energy Error
	WCET (ms)	Energy (J)	WCET (ms)	Energy (J)		
Broadcast	17.45	6.54E-03	17.78	7.94E-03	1.82%	17.63%
MP ADPCM	206.73	47.94E-03	203.44	41.10E-03	1.59%	14.26%
MP JPEG	86.00	28.81E-03	86.77	27.46E-03	1.15%	4.66%

Estimation errors of execution time are a bit higher, when compared to single processor algorithms, mostly due to the simple modeling of the interconnection latency. Energy consumption estimation is pretty close to single processor benchmark.

The broadcast algorithm is similar to the producer / consumer problem but here there is one producer and three consumers. As the packets are sent as broadcasts, all consumers receive the same packet. The execution time estimation error for this benchmark is around of 2% and the energy estimation error is around of 18%.

MP ADPCM application uses two processors, being one the encoder and other the decoder. The first processor encodes 100 ms of 16 bit samples of audio data, and sends over the network 100ms of 4 bit samples. The decoder inverts the process and decodes all 4 bit samples, placing the decoded 16 bit sampled audio in a buffer. The process is iterated ten times, so a one second audio sample is processed. In this benchmark, the execution time error is less than 2%, and the energy estimation error is around of 14%.

A processor of MP JPEG encoder acts like a master processor sending data blocks as broadcast image to the three others processors, which act like slaves. Each slave processors executes, concurrently, algorithms of DCT, quantization and zigzag scan on the samples. The results are sending back to the master that executes Huffman encoding and bitstream organization. The encoded image is placed into a buffer. This application had 1% of execution time estimation error and less than 5% of estimation error concerning energy consumption. This is not a big difference, compared to the single processed implementations of the last two algorithms.

Finally yet importantly, is important to state that VHDL simulation becomes unfeasible for a complete platform running complex distributed applications. For this case study, the VHDL simulation took about five days on a PC with a 2.8 GHz processor and 1 GB of RAM, on the longer algorithm. Nevertheless, our tool reported the results on just a few seconds.

6 Conclusions and Ongoing Work

This work proposes a methodology for software execution time and energy consumption estimations of homogeneous MPSoCs, which is composed by a design flow and a simulation tool.

To evaluate the design flow and calibrate the simulation tool, an MPSoC platform comprised by four Plasma processors cores, which communicate through an internal bus, was implemented and used as case study.

A gate-level simulation is used as reference to a higher-level simulation tool. Although we noticed that a small error in the estimations was introduced in the higher-level simulations of the case study architecture, the benefit from speedup in simulation time (which is orders of magnitude smaller, in the proposed tool) becomes quite attractive, as many different application scenarios can be evaluated in a short period.

The estimation tool is still under development stage.

Control structures, memory, processor registers, instruction counting and estimated energy consumption were replicated for full platform emulation.

The most complex structure of the estimation tool is the ISS. A great effort is being made to improve its precision. Such precision improvement tends to minimize errors in estimations of the whole platform, since the logic related to processing elements represents around 86.36% of global MPSoC logic.

References

[1] A. Hoffmann, O. Schliebusch, A. Nohl, G. Braun, O. Wahlen, and H. Meyr. *A Methodology for the Design of Application Specific Instruction Set Processors (ASIP) Using The Machine Description Language LISA*. **International Conference on Computer Aided Design (ICCAD)**, 2001.

[2] P. Mishra, M. Mamidipaka, and N. Dutt. *Processor-Memory Co-exploration Using and Architecture Description Language*. **ACM Transactions on Embedded Computing Systems**, v.3, n.1, pp.140-162, 2004.

[3] R. Leupers. *HDL-based Modeling of Embedded Processor Behaviour for Retargetable Compilation*. **International Symposium on System Synthesis (ISSS)**, 1998.

[4] Coware, Lisatek - www.coware.com/products/lisatek.php. Last access on Aug. 2007.

[5] ARM, Maxcore - www.arm.com. Last access on May 2005.

[6] D. Crisu, S. D. Cotofana, S. Vassiliadis, P. Liuha. *High-Level Energy Estimation for ARM-Based SOCs*. http://ce.et.tudelft.nl/publicationfiles/794_12_dcrisu_samos2003.pdf. Last access on Sep. 2007.

[7] P. Landman. *High-Level Power Estimation*. **International Symposium on Low Power Electronics and Design**, pp. 29-35, 1996.

[8] P. Giusto, G. Martin, and E. Harcourt. *Reliable Estimation of Execution Time of Embedded Software*. **Design Automation and Test in Europe (DATE)**, 2001.

[9] G. Bontempi and W. Kruijtzer. *A Data Analysis Method for Software Performance Prediction*. **Design Automation and Test in Europe (DATE)**, 2002.

[10] A. Muttreja, A. Raghunathan, S. Ravi, N. K. Jha. *Automated Energy/Performance Macromodeling of Embedded Software*. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v.26, n. 3, Mar. 2007.

[11] J. Bammi, E. Harcourt, W. Kruijtzer, L. Lavagno, and M. Lazarescu. *Software Performance Estimation Strategies in a System-Level Design Tool*. **International Workshop on Hardware/Software Codesign (CODES)**, 2000.

[12] Y.-T. Li and S. Malik, *Performance Analysis of Embedded Software Using Implicit Path Enumeration*. **32th Design Automation Conference (DAC)**, 1995.

[13] F. Wolf and R. Ernst. *Intervals in Software Execution Cost Analysis*. **13th International Symposium on System Synthesis, (ISSS)**, 2000.

[14] Mentor Graphics - www.mentor.com/products. Last access on Jan. 2008.

[15] César Augusto Missio Marcon. *Modelos para o Mapeamento de Aplicações em Infra-estruturas de Comunicação Intrachip*. **PhD Thesis, Universidade Federal do Rio Grande do Sul (UFRGS)**, 2005.

[16] Aldec - www.aldec.com/products/active-hdl/. Last access on Jan. 2008.

[17] Opencores - www.opencores.org.uk/projects.cgi/web/mips/. Last access on Jan. 2008.

[18] R. Fisher, S. Perkins, A. Walker, and E. Wolfart. *Sobel edge detector* - homepages.inf.ed.ac.uk/rbf/hipr2/sobel.htm. Last access on Oct. 2006.

[19] The International Telegraph and Telephone Consultative Committee. *ISO/IEC 10918-1 information technology digital compression and coding of continuous-tone still images requirements and guidelines*, 1992.