4544

A UNIFIED HARDWARE-SOFTWARE FRAMEWORK FOR EVALUATING POWER CONSUMPTION OF EMBEDDED SYSTEM-ON-A-CHIP DESIGNS

A DISSERTATION SUBMITTED TO THE GRADUATE DIVISION OF THE UNIVERSITY OF HAWAI'I IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN

ELECTRICAL ENGINEERING

DECEMBER 2004

By
Claudio Talarico

Dissertation Committee:

Vinod Malhotra, Chairperson
Tep Dobry
Lloyd H. Hihara
Nancy E. Reed
Jerzy W. Rozenblit
Galen H. Sasaki

To the memory of my grandmother

ACKNOWLEDGMENTS

This thesis would have not been possible without the help of many people. I wish to express my deepest gratitude to all of them.

Many thanks to my advisor Vinod Malhotra for his support and encouragement throughout the entire course of this work

Special thanks to Professor Jerzy Rozenblit for many technical discussions and above all for his friendship. This work could not have been completed without him.

Also, many thanks for their help to Ebi, Aseem, Jianfeng, Lizhi, Faisal, Qinglong, Haiyan, Liana, Abhay, Ying, Benhai, Dai, Steve, Ashok, and Ramesh.

I am deeply indebted to Klaus Buchenrieder, Ulrich Nageldinger, Helge Kleve, and Andreas Pyttel of the embedded system group at Infineon Technologies, Munich for both technical and financial support. It was a long conversation with Klaus Buchenrieder that inspired my interest in embedded systems. I also would like to thank Eberhard Escales and Reinhard Klinger of the automotive group at Infineon for their many helpful hints on microcontroller's power dissipation, and for promptly providing me hardware prototypes and various components.

"Grazie" to Giacomo, Tiziana, Costanza, Joe, Daniela, Laura, Jennifer, Frederick, Katerina, and Simone to make my stay in Honolulu a pleasant and memorable experience. Special thanks to Angie Hinrichs for painstakingly reading this thesis and giving me many helpful comments.

Finally, I want to thank the people closest to me, just for existing ☺.

Claudio Talarico

ABSTRACT

Recent years have seen a tremendous growth in both the complexity and demand for embedded computing systems. The rapid increase in design complexity combined with stringent time-to-market requirements has resulted in a need for computer aided design (CAD) tools that can help to make fundamental decisions as early as possible in the design cycle.

In this thesis we developed a framework to efficiently estimate power consumption of embedded System-on-a-Chip (SOC) designs at the system level. Most work to date has focused on the component level, rather than the system level. In these techniques, power consumption is estimated by associating to each component of the system a power model, which is obtained by pre-characterization either at the gate level or at the transistor level. Unfortunately, in order to perform a gate level or a transistor level characterization, a detailed knowledge of the components' internal structure is needed. At the early stage of the design cycle, such information may not be available or intellectual properties (IP) providers may not want to disclose it.

The major contribution of this thesis is to overcome this deficiency. We propose an estimation technique in which the power figures associated to each component of the system are derived from the execution of high level models rather than gate-level or transistor level pre-characterizations. The chief benefit is that design assessment can be done much earlier in the design cycle. As a consequence, engineers can be assisted in their decision-making process from the very beginning of the system design cycle. The use of a higher level of abstraction also leads to approximately three orders of magnitude

speedup of the estimation time. We have compared our system-level approach against gate level simulation and actual measurements on the real hardware. The results are within 10% of the gate level and physical measurement.

TABLE OF CONTENTS

ACKNOWLEDGMENTS iv

ABSTRACT v

CHAPTER 1: INTRODUCTION 1

1.1 Motivation 3

1.2 Low Power Design 4

1.2.1 Portable Computing Systems 4

1.2.2 Thermal Dissipation 5

1.2.3 Reliability 6

1.2.4 Environmental Considerations 7

1.3 Scope and main contribution of the thesis 8

1.4 Thesis Overview 10

CHAPTER 2: RELATED WORK 11

2.1 Sources of Power Consumption 11

2.1.1 Switching Power 12

2.1.2 Short-circuit Power 13

2.1.3 Leakage Power 14

2.2 Power Minimization Techniques 15

2.3 Low Power Design Techniques 18

2.4 Power Evaluation Approaches 19

2.4.1 Circuit Level 20

2.4.2 Logic Level 20

2.4.3 Register Transfer Level 21

2.4.4 System Level 24

2.5 System level Design Tools 26

CHAPTER 3: A UNIFIED HARDWARE-SOFTWARE DESIGN METHODOLOGY 29

3.1 Design Representations 30

3.2 Levels of Abstractions 31

3.3 Embedded SOC Architectures 32

3.4 A Framework for System Level Power Estimation 34

3.4.1 Processor Power Dissipation 40

3.4.2 Cache Power Dissipation 43

3.4.3 Main Memory Power Dissipation 49

3.4.4 Bus Power Dissipation 50

3.4.5 Peripheral Power Dissipation 53

3.4.6 Interconnect Power Dissipation 56

3.4.7 Off-Chip Power Dissipation 58

3.5 Summary 58

CHAPTER 4: PEACE: A FRAMEWORK FOR **P**OWER **E**STIMATION **A**IMED AT THE **C**ODESIGN OF **E**MBEDDED SYSTEM ON A CHIP 60

CHAPTER 5: EXPERIMENTS AND RESULTS 67

5.1 Peripherals Power Dissipation 67

5.1.1 Peripheral Illustrative Example 68

5.1.2 Setup and Results 70
5.2 Memory Power Dissipation 76
5.2.1 SRAM Illustrative Example 77
5.2.2 Setup and Results 78
5.3 Processor Power Dissipation 80
5.3.1 Processor Illustrative Example 80
5.3.2 CPU Power Characterization 82
5.3.3 Setup and Results 83
5.4 Summary 84
CHAPTER 6: SYSTEM LEVEL OPTIMIZATION 86
6.1 Motivation 87
6.2 Performance Evaluation 89
6.3 Multi-Criteria Decision Making Methods 91
6.3.1 The Weighted Sum Method (WSM) 92
6.3.2 The Weighted Product Method (WPM) 93
6.3.3 The Analytic Hierarchy Process (AHP) 93
6.4 A Design Example 94
6.5 Experimental Setup and Results 97
6.6 Summary 100
CHAPTER 7: CONCLUSIONS AND FUTURE WORK 101
7.1 Conclusions 101
7.2 Future Work 102

APPENDIX A: 103

BIBLIOGRAPHY 118

CHAPTER 1
INTRODUCTION

Recent years have seen a tremendous growth in both complexity and demand of embedded computing systems. A typical embedded system consists of a collection of programmable integrated circuits, such as microprocessors, microcontrollers, and digital signal processors, surrounded by application specific integrated circuits (ASICs) and standard components. The system interacts with the environment through sensors and actuators, while running a specific application program [1]. The range of applications for embedded systems is extremely broad and diverse and includes automotive, telecommunication, robotics, biomedical electronics and consumer electronics.

The rapid increase in design complexity, combined with stringent time-to-market requirements, have resulted in a need for computer aided design (CAD) tools that can help to make fundamental decisions as early as possible in the design cycle [2]. Unfortunately, the design methodologies currently used are still based on principles and tools that are not adequate for the complexity of the applications being developed. In order to cope with the complexity of present and future electronic systems, designs must be tackled at a higher level of abstraction and a unified framework for describing both hardware and software is highly desirable. A powerful paradigm to address this need is provided by the concept of "platform" [3]. A platform is simply an abstract view of the computing system. Designing a complex system may be regarded as the process of stacking up layers of such platforms. Depending on the extent of abstraction, namely the level of details used to describe the system, different concerns can be addressed and solved. The key is to model the system accurately at each level of abstraction with as few

details as possible. As design progresses, the system description can be refined with increasingly accurate and detailed information. At each layer, the goal is to collect quality/performance indices to assist the development team in making sound engineering decisions.

Among the many performance indices that can be used to characterize the quality of an embedded system design, power consumption has emerged has one of the most important. This is motivated by i) the proliferation of mobile computing devices, ii) the increased speed and density of CMOS VLSI circuits, and iii) the continuous shrinking of the transistor size using current deep submicron technologies. As the clock frequency and the silicon area increase, power consumption and the accompanying thermal heat dissipation are becoming severe problems for circuit designers. This is particularly true for mobile wireless devices, where lengthening battery life and thermal management are important criteria of the designs.

The aim of this work is to provide a framework to efficiently estimate power consumption of embedded System-on-a-Chip (SOC) designs at a high level of abstraction – the system level. We pursue the implementation of a system level approach that makes use of executable discrete event models [4]. Due to the generality of the modeling concepts adopted in this approach, the ideas introduced may be extended to other performance indices and may not be limited to just power dissipation. These may be used to address design space exploration, system synthesis optimization, and dynamic system management.

1.1 Motivation

It is important to address the power estimation problem as early as possible in the design flow. It is in the early stages that the impact of decisions is more critical to avoid expensive and time consuming design iterations. Most work to date has focused on the component level, rather than the system level. In these techniques, the power consumption is estimated by associating a power model to each component of the system. The power model is obtained by pre-characterization of the component either at the gate level or at the transistor level. Unfortunately, in order to perform a gate level or a transistor level characterization, a detailed knowledge of the components' internal structure and implementation is needed. At the early stage of the design cycle, such information may not be available at all or intellectual properties (IP) providers may not want to disclose it. Besides, the knowledge of power consumption for a given application provides little information about the power consumption of the same system for a different application. As a consequence, since characterization-based power models are accurate only if evaluated in very similar conditions as the one used for characterization, this may not be very useful for new designs and/or new applications.

The major contributions of this thesis are to propose and implement an approach that overcomes this deficiency. We propose an estimation technique in which the power figures associated with each component of the system are derived from the execution of higher level models rather than gate-level or transistor level pre-characterizations. As a consequence, the engineers are assisted in their decision-making process from the very beginning of the design cycle. An added benefit for the use of a higher level of abstraction is a noticeable speedup of the estimation execution time.

1.2 Low Power Design

As mentioned earlier, the need for low power design is motivated by several factors. These factors are described in further detail in the following sections.

1.2.1 Portable Computing Systems

The growing demand for portable computing systems is the primary cause that has led power dissipation to become such an increasingly important criterion in the design of CMOS VLSI chips [5]. Portable computing devices include notebook computers, personal digital assistants (PDA), palmtop computers, cellular phones and pagers, wireless modems and network cards, handheld videogames, music players, digital cameras, etc. Although very diverse in applications and designs, all portable systems share the common characteristic of being battery driven. Energy is supplied by an electrochemical battery, converting chemical energy into electrical energy. The battery capacity, that is the amount of energy the battery can provide, is limited. The time taken for the battery to become discharged is called lifetime or time to failure of the battery [6]. Power is the rate at which energy is consumed. The lifetime of the battery, together with its weight and costs, play a major role in the commercial success or failure of portable electronic products.

The trend in portable electronics is towards devices that can perform computational intensive tasks. Meeting increased computational requirements implies also increased energy requirements. Unfortunately, the advances in battery technology have not kept pace with the growth of energy consumption required for the devices the market demands. As a result, the need for low power design techniques and tools that can help to

minimize power consumption have become major design concerns. A typical Nickel-Cadmium (NiCd) battery provides energy densities of around 51 Watt-hours/kg [7]. More sophisticated chemistries such as Nickel-Metal Hydride (NiMH) allow a 20 to 30 percent density improvement [8]. Recently, the two battery technologies dominating the market are Lithium-ion and Lithium-polymer. The energy density achievable is around 150 Watt-hours/kg, which means that to provide 3 hours of operation to a device that consume 50W we need a battery weight of 1 kg (as a reference, a 0.13μm Pentium 4 operating at 2.2 GHz dissipates a typical average power of about 55 Watts [9]). Thus, despite the growth in battery capacity and the reduction in weight and cost, the improvements in battery technology alone are inadequate for the demand.

1.2.2 Thermal Dissipation

The electrical energy that an integrated circuit draws from its supply is transformed in heat. Unless this heat is removed, the operating temperature of the integrated circuit will increase [8].

$$T_j = T_a + \theta_p \cdot P_D \tag{1-1}$$

In this equation, T_j is the operating temperature of the integrated circuit (also known as junction temperature), T_a is the ambient temperature, θ_p is thermal resistance of the package, and P_D is the power consumed by the integrated circuit. As operating temperature increases, so does the propagation delay of the devices on the chip, and as a consequence, it becomes increasingly harder to meet the constraints on speed required for the desired behavior of the circuit. The timing characteristic of the devices on the chip are rated only for a specific temperature range, so exceeding the range can cause an

unpredictable behavior of the circuit. In addition, at higher temperatures the likelihood of failure due to physical phenomena considerably increases. It has been estimated that every 10 °C increase in the operating temperature will approximately double the failure rate of the circuit [8]. In order to maintain the operating temperature within the specified levels, it may be necessary to use expensive packages or some form of cooling mechanism. The result is a significant increase in the cost of the product. As clock frequency and silicon area increase so does the power dissipated.

$$P_D \propto \text{Die\_size} \cdot \text{clock\_frequency} \cdot V^2 \tag{1-2}$$

Thus, the current trend of integrating more functionality on a single chip while simultaneously increasing the computational speed of the circuit makes the problem of thermal dissipation even more critical.

1.2.3 Reliability

A higher power dissipation combined with shrinking transistor size exacerbates several reliability and signal integrity issues such as electromigration, resistive voltage drop, and hot carrier effect. Electromigration occurs due to the high current densities in the interconnect lines leading to movement of atoms and causing open or short circuits. Besides physical damage to the chip, high current density in the interconnects also causes significant resistive voltage (I·R) drops on the power supply lines. Due to I·R drop, portions of the chip may operate at a lower voltage than they have been designed for. The result is degraded performance and lower noise margin. Hot-carrier effect consists of charge trapping in the oxide or at the interface between semiconductor and oxide. This trapping phenomenon results in a shift in the transistor threshold voltage, and degradation

in transconductance and mobility [7]. As a result, the characteristics of the transistor are significantly altered with respect to their nominal values. In addition to high current density, also rapid current transition can cause problems. Excessive current variation in the supply line can lead to voltage fluctuations that affect the operation of the circuit. This is primarily due to the parasitic inductances of package pins and bonding wires of the integrated circuit. This problem is known as ground bounce or L·di/dt ringing.

Because of the direct relationship between electric current and power consumption, reducing power consumption has a beneficial effect on circuit reliability. The relationship between power and electric current is expressed as follows.

$$p = \frac{dw}{dt} = \left(\frac{dw}{dq}\right)\left(\frac{dq}{dt}\right) = v \cdot i \tag{1-3}$$

where p is the power consumption, w is the energy supplied, q is the electric charge, t is the time, v is the voltage, and i is the current.

1.2.4 Environmental Considerations

Environmental concerns are also an important motivation for low power design. According to the U.S. Environmental Protection Agency (EPA), approximately 80% of the total office equipment electricity consumption is due to computing equipment. In order to encourage the development of environment-friendly systems, in 1992 the EPA launched the "Energy Star Program" [10]. This resulted in the specification of standards and guidelines for manufacturers [7].

In 2003, according to the U.S. Department of Energy (DOE), because of the Energy Star Program, "Americans prevented greenhouse emissions equivalent to those from 14

million vehicles and avoided using the power that fifty 300-megawatt power plants would have produced, while saving more than $7 billion" [11].

1.3 Scope and main contribution of the thesis

This thesis deals with low power design of embedded computing devices at the system level. The driving force behind the new explosion of interest in embedded system design is the growing gap between the number of transistors that can be manufactured on a chip and the rate at which these transistors can be designed. This problem is know as the "design productivity gap" [12], and is quantitatively illustrated in Figure 1-1.

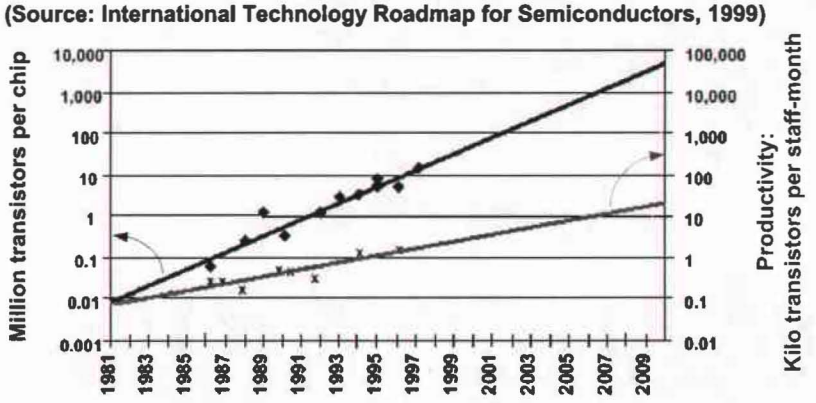

Figure 1-1 Design productivity gap

According to a Defense Advanced Research Projects Agency (DARPA) [13] report presented in 2002 and an even more recent statistics presented by Philips Semiconductor in 2004 [14] the trend shown in Figure 1-1 has not changed. The annual compound growth rate of design complexity remains approximately 59% while the annual compound growth rate of design productivity remains approximately 21%.

Due to their hybrid nature, in other terms the fact that they embody both hardware and software components, embedded systems offer an ideal solution for coping with complexity. The design of embedded systems has traditionally been carried out by decomposing and allocating the system to hardware and software, then allowing separate hardware and software design teams to design their respective parts, and finally integrating hardware and software. This separation of design tasks leads to the potential for initial design mistakes to be carried until the integration phase where they are much more difficult and costly to correct. We address this issue by using a unified framework for both hardware and software. The idea is to keep the hardware and software design activities tightly coupled.

This thesis proposes a new power estimation technique targeted toward the design of embedded systems. The distinguishing feature of this work is that power estimation can be done much earlier in the design cycle. The earlier the estimation step is performed in the design cycle the larger are the possibilities to reduce power consumption [7]. We developed a framework in which the power figures associated with each component of the system are derived from the execution of high level models rather than gate-level or transistor level pre-characterizations. The framework consists of four steps that lead to the overall system power. The steps are: 1) translating each component's functionality to a set of primitive instructions, 2) simulating the application program, 3) mapping the instructions requested by the application program into abstract functional units, and 4) computing the aggregate power consumption of the entire system.

The main benefits of performing power estimation at the system level are: 1) more compact design description, 2) faster execution time, and 3) earlier and faster design

exploration. The penalty is the potential loss of accuracy. The feasibility of a system level technique depends upon an appropriate tradeoff between accuracy and computation time.

1.4 Thesis Overview

The rest of this thesis is organized as follows. Chapter 2 describes previous work in power estimation. Chapter 3 introduces the basic tenets of our power estimation methodology. Chapter 4 describes its implementation details. Chapter 5 illustrates the methodology by applying it to different case studies. The results obtained by using our system level approach are compared against the results obtained by applying the gate level approach and the results obtained by performing direct measurement on the real hardware platform. Although the methodology presented address only power consumption, due to the generality of the conceptual models adopted, we can extend it to any performance metric. When several simulateneous performance metrics are considered, due to the fact that some of them may be conflicting, it is necessary to integrate the performance assessment framework with a specific process that allows the designer to select among the set of alternative designs explored the one that optimally satisfy all performance metrics involved. Chapter 6 discusses the use of multi-criteria decision making methods to rank the many alternative design instances of an embedded system. The various design instances are analyzed with respect to multi-objectives. The approach is illustated by applying it to the design of an aircraft pressurization system. The last chapter provides conclusions and discusses ideas for future work.

CHAPTER 2
RELATED WORK

In this chapter, a brief review of the main sources of power consumption in digital static CMOS circuits is presented, followed by a summary of the power evaluation techniques used in IC design.

2.1 Sources of Power Consumption

There are three sources of power consumption in digital static CMOS circuits: 1) switching power, 2) short circuit power, and 3) leakage power. Switching and short-circuit power are dynamic contributions, they depend on the behavior of the circuit. The leakage power is a static contribution. It is essentially considered to be a constant, and depends only on technology parameters, such as the area of the diffusion regions (drain and source of the transistors), and the area of the junctions between wells and substrate. The contribution of each of these sources to the average power consumption is summarized as follows.

$$P_D = P_{switching} + P_{short-circuit} + P_{leakage} \tag{2-1}$$

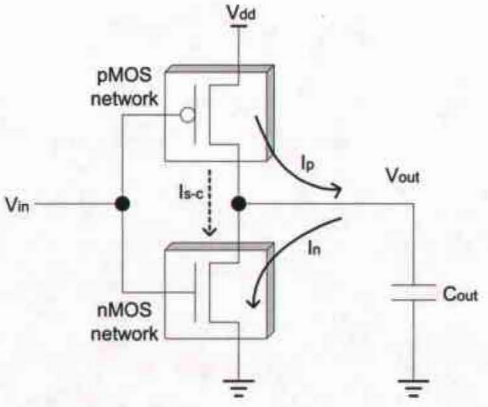

Figure 2-1 Power Dissipation in a static CMOS cell

2.1.1 Switching Power

The switching power is due to the flow of current occurring to charge and discharge the output capacitance (C_{out}) of the circuit. As the input of the cell transitions from high to low, the pMOS transistor network conducts while the nMOS transistor network is in cut- off. This creates a direct path between V_{dd} and C_{out}. As a result, a current I_p flows from the supply and charges up C_{out} to a voltage level of V_{dd}. As the input of the cell transitions from low to high, the nMOS transistor network conducts, while the pMOS transistors network cuts off. This creates a direct path between C_{out} and ground, leading to the circulation of current I_n, and the consequent discharge of C_{out}. The amount of energy stored or lost every time the output transitions is equal to:

$$E_{cout} = \frac{1}{2} \cdot C_{out} \cdot V_{dd}^2 \tag{2-2}$$

If we assume that the cell is part of a synchronous circuit operating at clock frequency f_{clk}, and we indicate with N the average number of transitions per clock cycle at the cell output (also known as switching activity), then we can represent the average switching power with the following equation:

$$P_{switching} = \frac{1}{2} \cdot C_{out} \cdot V_{dd}^2 \cdot N \cdot f_{clk} \tag{2-3}$$

C_{out} is a lumped model of the output load driven by the cell. The output load consists of three components:

$$C_{out} = C_L + C_{int} + C_{wire} \tag{2-4}$$

where C_L is the sum of the input capacitive load of the cells to be driven. The input load of a cell is the sum of the gate to source, gate to drain and gate to bulk capacitors of its

input transistors. The capacitance C_{int} represents the internal output capacitance of the cell. This consists of the capacitance of the diffusion regions connected to the output. The diffusion capacitance connected to the output is due to the source to bulk and drain to bulk capacitors of the cell itself [15]. C_{wire} is the parasitic capacitance of the interconnection between the driving cell and the driven cells. These are typically wire to substrate or wire to wire capacitances. Wires are constructed using either metal or poly-silicon.

The switching power is the largest contribution to the power consumption of a circuit. As a result, commonly, power estimation techniques focus only on this component. Switching power can be minimized by keeping the output load as small as possible by lowering the number of output transitions and by reducing the supply voltage. Since the dependency of the supply voltage on the switching power is quadratic, reducing supply voltage is the most effective option.

2.1.2 Short-circuit Power

The switching power equation does not consider the rise and fall times of the input and output signals. In reality, the input voltages applied to a cell have finite slopes. When the input voltage changes between 0 and $V_{dd,}$ during the interval of time for which the input voltage is within V_{tn} and V_{dd}-$|V_{tp}|$, both nMOS and pMOS transistors are simultaneously conducting. This creates a current path between V_{dd} and ground and as a result a short-circuit power dissipation. V_{tn} and V_{tp} are the threshold voltages of the nMOS and pMOS transistors respectively. The short circuit power is given by:

$$P_{short-circuit} = I_{s-c} \cdot V_{dd} \tag{2-5}$$

where $I_{s\text{-}c}$ is the average short circuit current flowing through the pMOS and nMOS networks. Assuming equal rise and fall times ($t_{rise}=t_{fall}\equiv\tau$) and symmetrical pMOS and nMOS transistors (threshold voltages $V_{tn}=|V_{tp}| \equiv V_t$ and transconductances $\beta n=\beta p\equiv\beta$) the short circuit power can be approximated as follows [15].

$$P_{short\text{-}circuit} = \frac{\beta}{12}\cdot(V_{dd} - 2V_t)^3 \cdot \tau \cdot f_{clk} \cdot N \tag{2-6}$$

where f_{clk} is the clock frequency and N is the average number of transitions per clock cycle at the output of the cell.

In summary, the amount of time that both pMOS and nMOS transistors are simultaneously conducting depends on the way the transistors have been designed (specifically, their transconductance βn and βp, and their threshold voltages V_{tn} and V_{tp}), how often the input signal toggles (namely the switching activity), and the slope of the input waveform. Thus, through careful design, the power contribution of short circuit current can be kept small enough to represent only a second order effect [15].

2.1.3 Leakage Power

Due to the fact that an MOS transistor is never really off, even when the circuit does not perform any operation, there are leakage currents flowing through the circuit. There are two sources of leakage currents, so the resulting leakage power consumption can be expressed as follows.

$$P_{leakage} = (I_{diode} + I_{subtreshold})\cdot V_{dd} \tag{2-7}$$

I_{diode} refers to the leakage currents due to the reverse biased parasitic diodes that form in the circuit. These diodes are between the diffusion regions (source and drain) and the

substrate, between the diffusion regions and the well, and at the junction between the well and the substrate. This current is usually very small, on the order of 25 μA for a million transistor chip [16] and is negligible compared to dynamic power consumption. $I_{subtreshold}$ occurs due to the non-zero diffusion of carriers between the source and the drain even when the MOS transistor is in cut off, indeed when the gate to source voltage (V_{gs}) is below the threshold voltage (V_t). In this region the MOS transistor behaves like a bipolar transistor and the subthreshold current depends exponentially on the difference between V_{gs} and V_t. Subthreshold current and the associated power consumption are usually negligible, however they can become significant for technologies based on low threshold voltages [17].

2.2 Power Minimization Techniques

In a well designed digital CMOS circuit, switching power is the dominant component of the total power consumption. Thus, optimizing power consumption becomes the task of reducing one or more of the parameters that determine switching power while maintaining the required functionality [17]. As summarized in equation (2-3), switching power depends on clock frequency (f_{clk}), power supply voltage (V_{dd}), switching activity (N), and output capacitance (C_{out}). It is important to realize that these parameters are not independent. In order to minimize power consumption the interaction between these parameters must be carefully considered.

Clock frequency affects power consumption, and it is also a measure of system performance. In IC design, trading speed for power dissipation is rarely a viable alternative [7]. In addition, it should be noted that a power saving does not necessarily

correspond to an energy saving. If a system performs a task consuming half the power it did before, but it also takes twice as long to complete, the energy consumed to achieve the task does not change.

Due to the quadratic dependence between supply voltage and switching power, "voltage scaling" has become the most popular solution to lower power. However, the term "voltage scaling" does not merely mean to lower V_{dd}. In fact, reducing only V_{dd} would also cause a reduction of the circuit speed. Decreasing V_{dd}, while keeping geometry and technology of the MOS transistors constant (that is, maintaining the current provided by the circuit constant), causes a reduced voltage swing (dv_{out}) and as a result a longer time (dt) is required to charge or discharge the output load capacitance (C_{out}) driven by the circuit. This mechanism is illustrated in Figure 2-2 and is analytically summarized by equation (2-8).

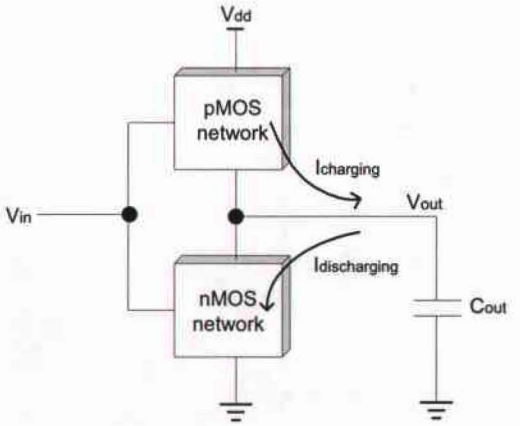

Figure 2-2 Static CMOS cell driving a load

$$i(t) = C_{out} \cdot \frac{dv_{out}(t)}{dt} \qquad (2\text{-}8)$$

The solution is to scale all device voltages and device dimensions (both horizontal and vertical) by the same factor k>1, so that the electric field remains unchanged and so that the power scales down by k^2 and the speed goes up by k [18]. There is a limit on the extent of voltage scaling permissible. The limit is imposed by the fact that reducing V_{dd} also reduces noise margins, and the delay is affected by the following relationship [19]:

$$\text{delay} \propto \frac{V_{dd}}{(V_{dd} - V_t)^2} \tag{2-9}$$

As V_{dd} approaches V_t the speed of the circuit degrades drastically. A common rule of thumb is to keep V_{dd} larger than $4V_t$ [20]. As suggested by equation (2-9), the performance penalty due to scaling the power supply voltage (V_{dd}) can be minimized by reducing the threshold voltage (V_t). However, decreasing V_t beyond a certain limit leads to a significant increase in the subthreshold leakage current, and the corresponding leakage power. A common work around is to dynamically vary the threshold voltage of the devices, using lower thresholds when the devices are active, and higher thresholds when they are not.

Another effective method of minimizing power consumption consists of reducing switching activity and output capacitance [21]. The product of switching activity and output capacitance ($N \cdot C_{out}$) is sometimes called effective capacitance or switched capacitance. The output capacitance can be minimized by: 1) reducing the amount of logic used for the desired functionality, 2) using smaller geometries, and 3) shortening the interconnections. The switching activity can be reduced by: 1) selectively shutting down parts of the circuit that are not active, 2) using clock gating techniques, 3) minimizing the number of bit transitions for the most likely states, and 4) encoding

consecutive data in order to reduce Hamming distance[22]. In contrast to voltage scaling, where power consumption is minimized through a significant intervention at the physical level, power consumption may be reduced by reducing the switching capacitance at all levels of the design hierarchy.

2.3 Low Power Design Techniques

In recent years, the complexity of the applications demanded has become so high, that the power savings obtainable through voltage scaling and "ad hoc" circuit techniques are no longer sufficient. Higher level strategies must be exploited. Figure 2-3 summarizes the power saving opportunities at the different level of the design hierarchy. The margin for power optimization is significantly larger at higher levels of abstraction. This is motivated by the fact that at higher level of abstractions many more tradeoffs can be explored and compared [23]. Figure 2-4 illustrates some of the possible venues for minimizing power consumption at different levels of design abstraction.

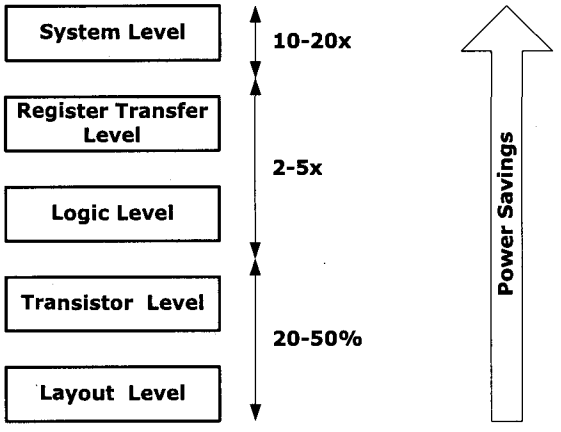

Figure 2-3 Power saving opportunities [23]

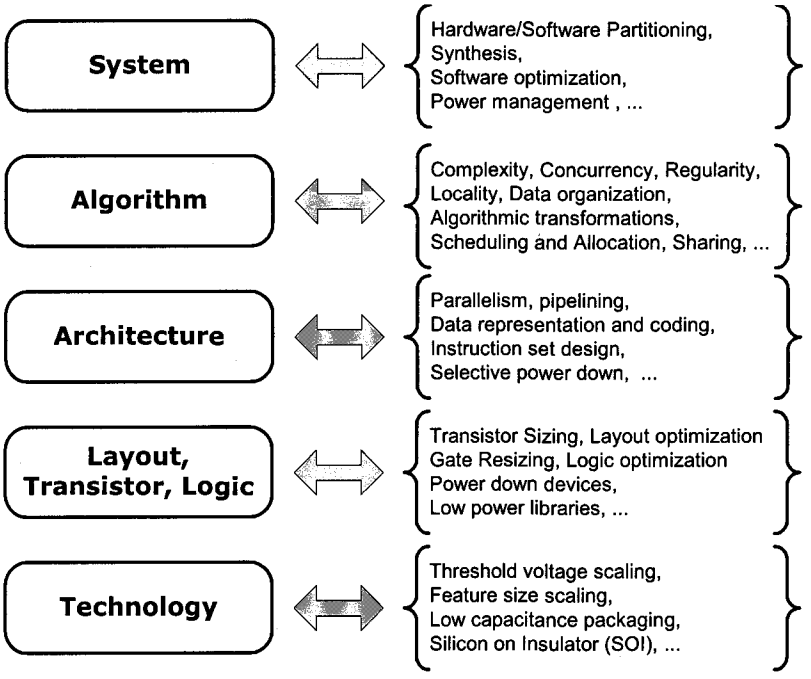

Figure 2-4 Approaches for minimizing power consumption [23]

2.4 Power Evaluation Approaches

Extensive research has been conducted on power evaluation over the past few years. Power evaluation can be done at several abstraction levels: Circuit-Level, Logic-Level, Register-Transfer-Level, and System Level. Different evaluation approaches may be better suited to different parts of a design or at different stages of the design process. As we move from the lowest to the highest abstraction level, accuracy of the estimation is traded off for speed. Figure 2-5 illustrates the trade off between accuracy and execution speed.

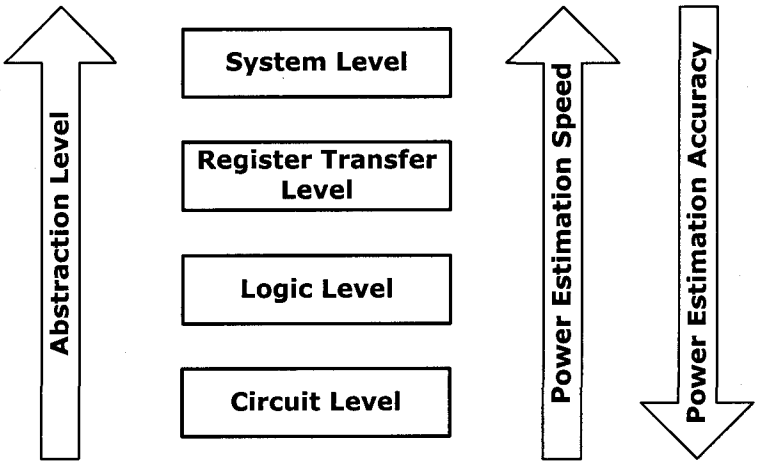

Figure 2-5 Accuracy and speed versus abstraction level

2.4.1 Circuit Level

Circuit level approaches simulate the circuit at the transistor or at the switch level and monitor the supply current [24]-[26]. Although these approaches provide excellent accuracy, they are not a viable solution for large designs. The execution time is too high. It would require weeks to evaluate the power consumption of an average SOC [27].

2.4.2 Logic Level

Logic level (or gate level) approaches simulate a design at the logic gate-level, and calculate power by considering the switching activity and the capacitance of nodes in the design [28]. Logic-level approaches execute orders of magnitude faster than circuit-level approaches but at the expense of accuracy. Even so, logic level approaches are still too slow. It would require days to evaluate the power consumption of an average SOC [29].

2.4.3 Register Transfer Level

RTL approaches are based on modeling the power consumption of more abstract components, such as muxes, adders, multipliers, registers, etc. RTL power estimation approaches can be classified in the following categories: analytical techniques and empirical techniques [5].

Analytical power modeling techniques compute power consumption based on estimates of design complexity and switching activity. There are two classes of analytical methods: complexity-based and information-theoretic-based techniques. Complexity based methods rely on the observation that a chip's complexity can be roughly estimated in terms of "gate equivalents". In effect, complexity can be specified in terms of the approximate number of reference gates that would be required to implement the design. In order to estimate the power consumption of a system, we simply multiply the number of equivalent gates by the average power consumed by the equivalent gate. An example of this technique is given by the Chip Estimation System (CES) developed in [30].

$$P = \sum_{i \in \{FU\}} GE_i \cdot \left(E_{typ} + C_L^i \cdot V_{dd}^2\right) \cdot f \cdot A_i \qquad (2\text{-}10)$$

where GE_i is the gate equivalent count for functional unit i, E_{typ} is the average energy consumed by an equivalent gate when active, $C_L{}^i$ is the average capacitive load per gate including fan-out and wiring, f is the clock frequency, and A_i is the average percentage of gates switching each clock cycle within the functional unit. Unfortunately, estimating power consumption by using the energy consumption of a single reference gate while neglecting different circuit styles, clocking strategies, and the specialized nature of certain blocks, leads to rather inaccurate approximations. Accuracy can be improved by

applying customized estimation techniques to different design components such as logic circuit, clock distribution network, memory, interconnect, and off chip drivers [31]. The main advantage of complexity-based methods is that they require very little information - a few technology parameters are usually sufficient for power estimation. However, due to the limited information available on how the application stimulates the inputs of the system, the accuracy with which circuit activity is modeled is not satisfactory. Typically, an overall fixed activity factor is assumed. Information-theoretic approaches try to solve this deficiency by estimating average activity and capacitance of a logic block based on the entropy of their input and output signals. The underlying idea is to use area as a measure of physical capacitance and entropy as a measure of activity [2],[32].

$$P \propto \text{Capacitance} \times \text{Activity} \propto \text{Area} \times \text{Entropy} \tag{2-11}$$

Given a random boolean variable x, its entropy is defined as:

$$H(x) = p \cdot \log_2 \frac{1}{p} + (1-p) \cdot \log_2 \frac{1}{(1-p)} \tag{2-12}$$

where p is the probability of x being 1, and 1-p is the probability of x being 0. Assuming that the input signals switch every clock cycle with probability 0.5, the area of a block, is related to the number of boolean inputs n, and the total entropy of its m outputs Hout as follows [33]:

$$\text{Area} \propto \begin{cases} \frac{2^n}{n} \cdot H_{out} & \text{as } n \to \infty \\ 2^n \cdot H_{out} & \text{for } n \le 10 \end{cases} \tag{2-13}$$

It has been proved [32] that the average entropy of all nodes x_i in a functional block depends on the entropy of its inputs (H_{in}), the entropy of the outputs (H_{out}), and the number of inputs (n) and outputs (m):

$$\text{Entropy} = \frac{2/3}{n+m}\left(H_{in} + 2 \cdot H_{out}\right) \quad (2\text{-}14)$$

Therefore, power estimation consists of running RTL simulations to measure the input and output entropy of all functional blocks in the design and then using equations (2-11)-(2-14) to derive power consumption.

Two alternative approaches have been proposed in ref. [34]. Here, input and output entropy are obtained by using closed-form formulae rather than running RTL simulations. The first approach computes entropy based on design structural information such as number of inputs, number of outputs, number of internal nodes, number of logic levels, and distribution of nodes on each level. The second approach is based on design functionality. The notion of entropy transfer function (HTF), defined as the fraction of information that is transferred though a function, is introduced and used. Since, for a given design, power consumption varies significantly with the application executed, the main limitation of analytical methods lies in the difficulty of capturing the correlation between the application and the attributes used for the estimation.

Empirical power estimation techniques measure the power consumption of existing implementations and produce models based on these measurements. There are two classes of empirical methods: characterization–based macro modeling and fast synthesis-based modeling. The characterization-based macro modeling technique consists of characterizing a lower-level implementation of the various RTL macro-blocks in the system under various "training" input sequences. Based on the data collected, a macro model is built. The macro-model describes the power consumption of the block as a function of its input and output statistics. The macro model can be either expressed in an

analytical form representing the curve that best fits the collected data or through the use of look-up tables [35]. An important issue limiting this approach is the potential bias due to the training data. The fast synthesis-based methods consist of performing a limited but faster synthesis pass on the RTL description. The design is mapped in a "meta-library" composed of a much smaller set of primitive cells than the complete technology library. The resulting netlist is used for power estimation either through gate level simulation or through probabilistic techniques.

The accuracy of power estimation through RTL approaches is satisfactory, typically within 5% to 10% of gate-level power estimates. Although the technique's computational time is orders of magnitude smaller than logic-level approaches [36],[37] RTL approaches are still too slow when applied to large designs.

2.4.4 System Level

System level approaches estimate power based on simple high-level descriptions of the system and the intended application. Conceptually, the ideas exploited are very similar to the ones introduced at the register transfer level, but here the granularity of the components considered is much higher. At this level, the components chosen as primitives are intellectual property (IP) cores such as microprocessors, microcontrollers, digital signal processors, main memory, caches, peripherals, and various types of interfaces. Thus, as we move up in abstraction the estimation becomes more and more difficult. An abstract notion of capacitance and switching activity is required to estimate power.

A typical embedded SOC contains both hardware and software components, so both aspects must be considered. Extensive research work has been conducted and different approaches have been proposed.

Software power consumption for microprocessor and DSP cores has been addressed, modeling power consumption mainly at the instruction level [8],[38]. Given a program execution trace, energy is computed as the sum of the energy consumed by each instruction executed. Energy consumed depends on the specific instruction being executed as well as on the previously executed instructions and on the data on which the instruction operates. This approach can be speeded up by deriving a trace file of reduced size but generating equal power dissipation [39]. A mathematical model of a generic 32-bit processor, obtained through functional decomposition of the activities carried out, is proposed in [40]. Instructions are classified based on the functional units exercised. The aim of the model is to estimate the static power consumption of the instructions executed, but it does not take into account that power consumption change significantly depending on the actual input data applied. An approach similar to the one used in [8],[38],[39] for processors has been proposed to estimate power consumption of peripheral cores [29],[41]. System level models for cache, memory, and bus power consumption have also been proposed in the literature [42]-[49]. These models consist mainly of closed form equations that express power consumption as a function of usage/traffic and the component parameters.

A relevant issue in the approaches considered so far is that they analyze performance and energy consumption of each core separately and then compute the final system performance and energy consumption by summing the result of each analysis, in other

words a constant additive model is assumed [50]. In practice, there are tight interdependencies between the various system cores. Since core interactions directly affect both performance and energy consumption, an integrated approach is needed for obtaining accurate estimations. The whole system has to be simulated as a unit. A first attempt to analyze how power consumption is affected by the interdependency between CPU, memory and cache is presented in [51]. However, since the impact of buses is not considered, the approach may experience a significant loss of accuracy in the case of deep sub-micron technologies. The interdependency between cache and buses can affect power consumption estimation significantly and it is discussed in [52].

A drawback regarding all system-level approaches examined is that the energy consumption models associated with each core are based on characterization done either at gate level or transistor level. In order to perform a gate level or a transistor level characterization, a detailed knowledge of the components' internal structure and implementation is required. Such information may not necessarily be available at the earliest stage of the design process or IP providers may not want to disclose it. Besides, the knowledge of power consumption of a system for a given application provides little information about the power consumption of the same system for different applications. As a result, characterization-based power models are highly accurate only if evaluated in the same conditions as the one used for characterization.

2.5 System Level Design Tools

Recently, there has been considerable effort in designing tools that enable system level assessment of different design performance metrics. Tools have emerged from both

the academic community and the industry. The pioneering work of Tiwari et al. [53] on system level power evaluation has resulted in a tool called Wattch. Wattch is a framework for analyzing and optimizing power consumption at the system level. It focuses on the microprocessors and it is built on top of the SimpleScalar simulator toolset developed at University of Wisconsin [54]. More recently, Henkel et al. [51] have introduced a power estimation and optimization framework called Avalanche. Avalanche is based on a generic embedded system architecture consisting of CPU, custom hardware, and memory hierarchy. The tool can be used either for simple design space exploration aiding designers to make more informed choices or in conjunction with optimization strategies. The input of the framework is an application program and is fed into the instruction set simulator of the target processor and into the memory trace profiler QPT [55], which in turn generates the memory access trace to be used by dineroIII [55] to predict the number of demands and misses in cache. Another interesting tool called Platune has been developed by Givargis et al. [56]. Platune can be used for power and performance tuning of SOC parameterized platforms. This tool represents one of the first attempts that focus not only on the processor and memory hierarchy but also on the peripheral components. In order to be economically viable a SOC must be general enough to be usable across several different applications. Therefore, SOC platforms are usually designed in a parameterized form and then optimally configured for the application at hand. The goal of Platune is to efficiently explore the many possible configurations by assigning values to different parameters, and computing the Pareto-optimal configurations with respect to power. The only commercial tool currently available has been developed by ChipVision in cooperation with OFFIS at University of

Oldenburg in Germany and is called Orinoco [57],[58],[59]. Orinoco is a tool suite for power estimation and optimization of design represented at the system level using SystemC, C/C++, or VHDL. The underlying power models used for power estimation are based upon accurate gate level simulation and statistical abstractions of a set of representative primitive blocks.

CHAPTER 3
A UNIFIED HARDWARE/SOFTWARE DESIGN METHODOLOGY

Traditionally the design of embedded computing systems has been carried out by decomposing and allocating the system to different hardware and software components. Figure 3-1 summarizes the main steps "traditionally" performed for designing embedded systems. This clear separation between hardware and software domains is at the core of several inadequacies [60]. Because hardware and software are developed using different methodologies and tools, it is extremely hard to verify the system in its entirety, so the system as a whole is usually not verified at all. Typically, verification is conducted in two phases. First, hardware and software are verified separately. Second hardware/software integration is verified. Because of this, potential incompatibilities at the hardware/software boundaries are carried through the integration phase, where they are much more difficult and costly to correct. Given certain system functionality, because hardware and software are considered separately, the architecture chosen is unlikely to be optimal. Furthermore, any change in the way hardware and software components are partitioned requires considerable redesign, modifications may negatively impact 'time to market'.

A unified view of hardware and software is needed to cope with these inadequacies. Adopting a unified view, hardware and software simply become two "implementation options" available to the designer to pursue the desired performance and functionality [61], rather than two different, often conflicting, domains. Three factors are fundamental to make such unified view possible: IC capacity, software compilers, and synthesis tools for hardware.

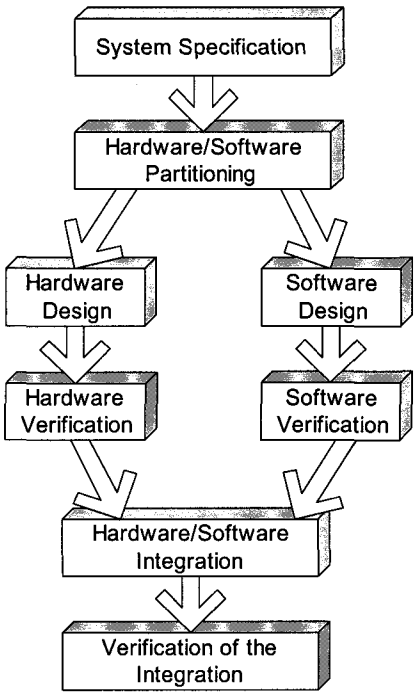

Figure 3-1 Traditional embedded system design flow

3.1 Design Representation

The choice of a unified representation for both hardware and software transform the problem of designing an embedded system into the task of finding a suitable manner of describing the processing behavior and also the different constraints and performance requirements of the system. The key to successfully describing behavior, constraints and objectives of a system consists of formalizing them through a syntax that can capture their semantics precisely and unambiguously. In order to achieve this goal we can rely on several mathematical models of computation (MoC). Some examples of commonly used MoC include finite state machines, extended finite state machines, data flow networks, discrete event systems, and Petri nets [62], [63]. Some MoC describe certain behaviors

more naturally than others, so it is important to be able to support a large number of MoC. Systems that are best described with two or more MoC are called "Heterogeneous" systems. The elements carrying out computations are called "processors" and they can be either "hardware" or "software" [61]. Since a MoC is just a conceptual notion, we need a language to capture that concept in a concrete form. Certain languages may be better than others at capturing different MoC. Languages can be expressed either in a textual or graphical form (e.g., state charts and graphs). However, defining a language in a graphical or textual form is completely equivalent, so the choice of a textual language versus a graphical language is independent from the choice of the MoC.

3.2 Levels of Abstraction

The key to successfully designing an embedded system is to identify the appropriate abstraction level for representing the system. Depending on the abstraction level used, indeed the amount of details, different concerns can be addressed and solved. Minimizing the number of details used is also crucial in order to minimize the computational time required to simulate and evaluate the system behavior. More accurate and detailed information can be added as the design evolves [64]. In effect, the design process consists of stepwise refinements of executable models representing the system behavior. The major benefits of keeping the abstraction level as high as possible are: 1) design assessments can be done earlier in the design cycle, 2) the time required to explore different design tradeoffs is shorter, and 3) the description of the system remains independent of implementation until the very end. There are two powerful paradigms to pursue this strategy: platform-based design and model-based design. A "platform" is

simply a formal description of the system and its components. Model based design consists of representing the useful properties of a platform as executable models specifying the system behavior, and then from these specifications synthesizing implementations. The underlying observation is that by using a modeling language to state all the important properties of a design, one can later refine the model into an implementation [65]. In effect, model-based and platform-based designs are two views of the same thing [66]. The term platform-based design is usually used to emphasize refinement efforts toward the physical implementation domain, while the term model-based design is usually used to emphasize refinement efforts toward the problem domain, and therefore, represents a more abstract view.

3.3 Embedded SOC Architectures

In general, embedded SOC architectures are formed by the following components: processors, memory hierarchy, peripherals, communication buses, and custom hardware parts. A generic embedded SOC architecture is depicted in Figure 3-2.

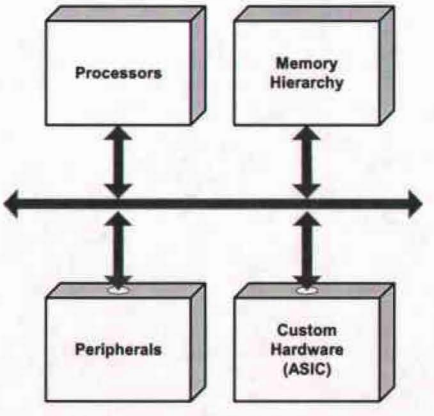

Figure 3-2 Generic embedded SOC architecture

Processors can be either general purpose, e.g., central processing units (CPU) and microcontrollers, or special purpose, e.g., digital signal processors (DSP). Memory hierarchy comprises different levels of main memory, instruction and data cache. Main memory can be either dynamic random access memory (DRAM) or static random access memory (SRAM). Peripheral devices can be extremely diverse both in nature and role. Their main function is to facilitate the interfacing of the SOC with the external environment. A few examples include universal asynchronous receiver-transmitter (UART), pulse width modulators (PWM), timers, universal serial bus (USB), direct memory access (DMA), analog to digital converters (ADC), and digital to analog converters (DAC). Communication buses permit the various blocks of the system to exchange both control and data information. A bus can be global or local, depending on the number of blocks that it interconnects, and external or internal depending on whether it is directly accessible by the external environment or not. Buses can be organized hierarchically. Custom hardware parts perform very specific and high performance tasks. Therefore, they are usually implemented with application specific integrated circuits (ASIC).

Keeping in mind that the aim of this work is the assessment of an embedded system's power consumption, the first step towards this goal is to identify techniques that allow one to effectively estimate power consumption of different classes of components that form an embedded system.

3.4 A Framework for System Level Power Estimation

Power estimation requires the capability of estimating the average switching activity and the total capacitance of a circuit:

$$P \propto A_{avg} \cdot C_{tot} \propto A_{avg} \cdot Area \cdot C_{avg} \tag{3-1}$$

where A_{avg} is a measure of the average node switching activity, C_{tot} is the total capacitance, C_{avg} is an estimate of the average logic gate capacitance for a given technology, and Area is an estimate of the area complexity of the circuit. Usually area complexity is expressed in terms of gate count rather than physical area. A measure of complexity based on gate count has the advantage of being independent from the target technology. As discussed in ref. [2], the power dissipation of a static CMOS circuit composed of N logic gates, whose gate output nodes are x_i, i=1,2,…,N, is given by:

$$P = \frac{1}{2} \cdot V_{swing}^2 \cdot \sum_{i=1}^{N} C_i \cdot A(x_i) \leq \frac{1}{2} \cdot V_{dd}^2 \cdot \sum_{i=1}^{N} C_i \cdot A(x_i) \tag{3-2}$$

where $A(x_i)$ represents the switching activity of node x_i, (average number of logic transitions per second of the node), and C_i is the total capacitance at node x_i. This expression accounts only for the switching component of dynamic power. This is the power dissipation due to the charging and discharging of the logic gates' output capacitance. The short-circuit component of dynamic power dissipation (power dissipation due to transient current flowing from power supply to ground in the unloaded logic gate as a result of the finite slope of the waveform applied at the logic gate's input), is neglected. This is motivated by the fact that typically short circuit power is only around 10% of the total power.

Since estimation techniques at the system level cannot rely on information about the gate-level structure of the various system components, an abstract notion of physical capacitance and switching activity must be introduced in order to predict power dissipation.

Informally, we define the power evaluation problem as follows: given a SOC architecture composed of several interconnected computational, communication and memory "cores", we have to devise a high level executable model of the system that can output power dissipation information during a system level simulation. We describe each core's power behavior using FSM as the underlying conceptual model of computation. Each state has a value associated with it representing the power dissipated in that state. Interdependency between system components is enclosed in the interaction between the state machines. Given a core *C*, its instructions $i_1, i_2, i_3, \ldots, i_n$, and its power states m_1, m_2, m_3, ..., m_k, we must implement a functional model of *C* that when executed enables power evaluation.

Although we focus on power consumption, the proposed approach can be extended to any other performance metrics. In order to effectively model a complex system we represent each component only with the minimum amount of information strictly needed to describe the behavior of interest. For example, if we are only interested in power consumption, all information regarding system functionality can be removed so that the model execution can be made considerably faster. The approach we propose tackles the problem of performance estimation at a very high level of abstraction. We generalize and extend the schema given in ref. [29]. The distinguishing feature is that we do not need to

rely on gate level simulation to characterize the "per instruction" power consumption of each core component of the system.

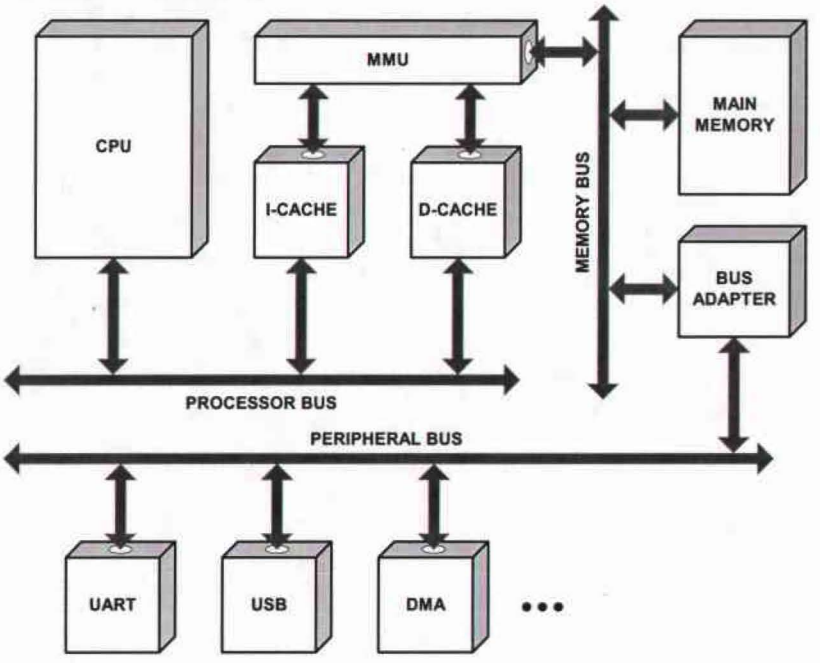

Figure 3-3 Embedded SOC's example

The total power dissipated by the system is approximated by:

$$P = P_{cpu} + P_{mem} + P_{caches} + P_{buses} + P_{offchip} + P_{interconnects} + P_{peripherals} \quad (3\text{-}3)$$

Evaluation of power consumption for custom hardware requires the same "ad hoc" technique needed for the peripherals, therefore, from a perspective of power consumption, peripherals and custom hardware are "classified" together.

Given an application program and an embedded SOC's target architecture our approach to estimate system power dissipation is based on the following constituents:

1. A compiler in order to map the application program to the SOC architecture,
2. Power models for each of the SOC components,

3. Tightly coupled simulation model for each of the SOC components. Simulation models collect and compute switching activity during a simulation run.

The application program, is typically written in languages such as C, C++, or Java. The use of the same high-level language for describing the power behavior of both hardware and software makes the modeling task much easier. The availability of SystemC makes the C++ language an ideal match to address the need of a unified hardware/software framework. SystemC is a modeling platform consisting of C++ class libraries and a simulation kernel for designing at the system and register transfer level [68]-[70].

Our approach is based on two key observations. First, we note that the behavior of a core can be seen as the execution of a sequence of "instructions", where the term "instruction" is used as a synonym of "action" and it is not necessarily expected to be atomic. Second, since during the early stages of the design process the gate level representation of most of the cores may not be available, we compute power dissipation analytically combining the technology parameters obtainable from data sheets with the activity information that we can gather by executing the core's high-level model.

We estimate power consumption in four steps. The first step consists of breaking each core's functionality into a set of instructions. The functionality of a component represents all possible behaviors that the components can assume; where we define behavior as an action or a set of actions that the component performs during the execution of an application. The major benefit of this step is that the complexity of the core's internal implementation is hidden behind the relatively simpler interface offered by the instruction set. However, there is a tradeoff in selecting the right set of instructions. Having many fine-grained instructions may lead to greater accuracy but longer

simulation time than having fewer coarse-grained instructions [29]. Notice that we associate with each core's instruction set only the information needed to describe the performance metrics of interest, for example, power consumption. The second step consists of simulating the application program and extracting a trace file for the core. A trace is defined as the sequence of instructions/data items that are executed by a core during its simulation. The aim is to estimate the switching activity of the core. The third step consists of mapping the instructions requested by the various tasks performed by the core into abstract functional units that are used to estimate complexity, such as the gate count. Given switching activity and complexity we can compute the core's power per instruction. The last step consists of connecting all the core models to compute the aggregate power consumption of the entire system.

In order to estimate the power consumption of a generic embedded system, we propose the framework illustrated in Figure 3-4. The underlying assumption is that any embedded system can be decomposed in three sections: processors section, memories section, and peripherals section. The framework is intended as a generic wrapper around the embedded system components. Each component has an associated simulation model and a monitor. The purpose of the monitor is to observe the execution of the simulation model and to probe the data needed to characterize the behavior of the component. Such data is then processed by the analyzers in order to compute the quality/performance metrics of interest. A distinguishing feature of the framework is its modularity. Such modularity helps to isolate the various system components from each other and to abstract from their implementation details. The major benefit is that the assessment of a design can be carried out from the earliest stages of the design cycle.

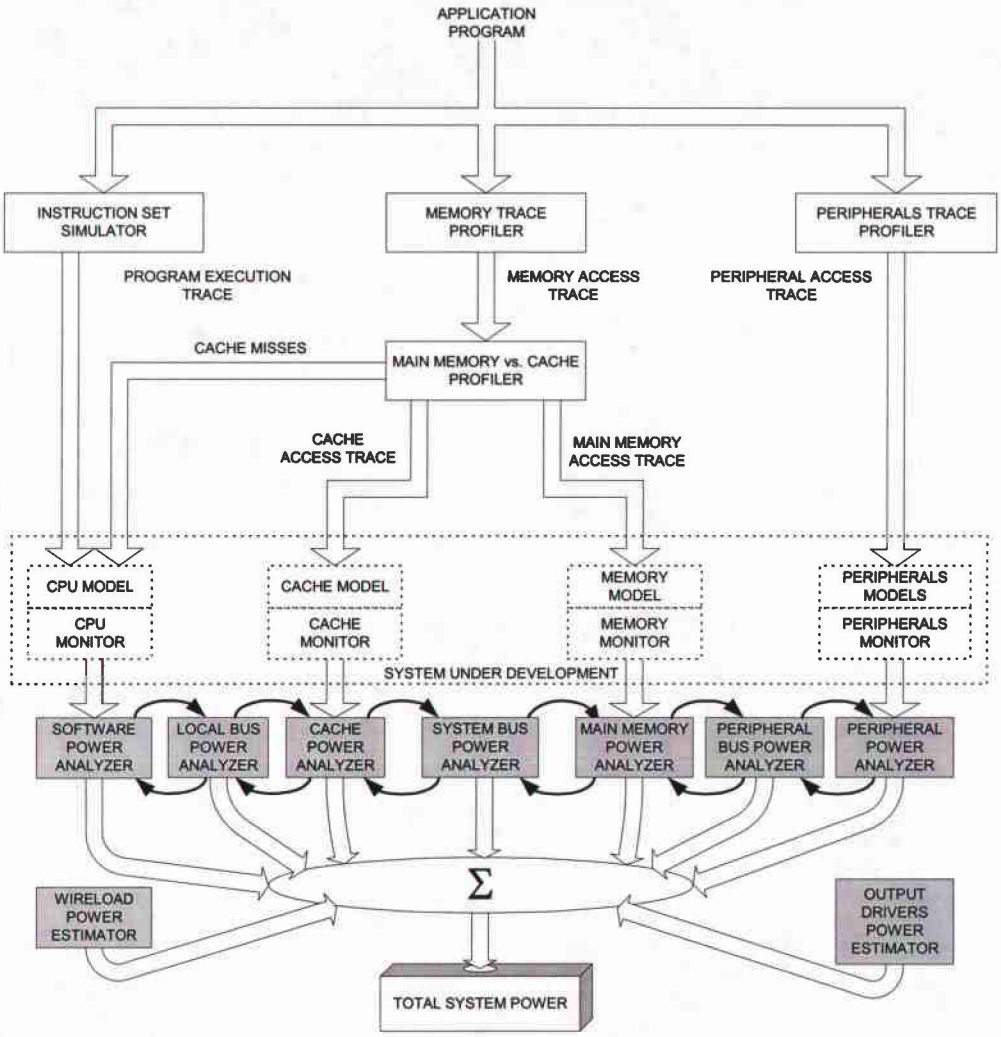

Figure 3-4 Power Estimation Framework

The input of the power estimation flow is the application program. The application program is fed into the instruction set simulator (ISS) of the target CPU to produce a program trace. The program trace is then post-processed by a software power analyzer. Its function is to estimate the power consumed by the processor as a consequence of the software executing on it. The application program is also fed into a memory trace profiler, to record all memory access traces. The memory access traces are then further

profiled to provide the number of cache demand misses for both data and instructions. This information is used by the software power analyzer to account for the additional power consumption due to the stalls caused by the cache misses. Such information is also needed by the main memory power analyzer and by the cache power analyzer to obtain the actual number of main memory and caches accesses, in order to compute their power consumption. Depending on whether peripherals are accessed through memory mapped I/O or dedicated I/O, a peripheral access trace can be extracted either from the memory access traces or from the program traces. Any access to or from main memory, caches, and peripherals translates eventually into information traffic over the communication buses. In order to compute the buses' power consumption, a specific bus analyzer is defined for each bus present in the system.

In deep sub-micron technology, the downscaling of the transistor feature size makes the relative importance of the power consumption of interconnecting wires and off-chip drivers comparable with the power consumption of the logic gates. The power dissipation of interconnections and off-chip drivers is computed based on physical estimations and technology parameters. Each of the power analyzer modules embodies the analytical expressions needed to compute the power consumption of the various types of cores.

The computation of power dissipated by each component in the system is discussed in the following sections.

3.4.1 Processor Power Dissipation

In order to estimate the power consumption due to the execution of the application software on the CPU, we rely on an ISS. The ISS maintains detailed statistics of the

processor's internal activity (fetches, stalls, instruction execution frequency, internal register accesses, and so on). Such statistics can be post-processed to compute power consumption. The technique used is an extension of the instruction-based approaches given in references [8],[29],[38], and [51]. The idea behind instruction-based approaches is that "by measuring the current drawn by the processor as it repeatedly executes certain instructions, it is possible to obtain most of the information that is needed to evaluate the power cost of a program for that processor" [38]. The average power dissipated by a processor running a certain program is given by:

$$P_{cpu} = I_{cpu} \cdot V_{dd} \tag{3-4}$$

Since power represents the rate at which energy is consumed, energy consumption is given by:

$$E_{cpu} = P_{cpu} \cdot T_{exec} \tag{3-5}$$

where T_{exec} is the execution time of the program and it is given by:

$$T_{exec} = N_{cyc} \cdot T_{clock} \tag{3-6}$$

where N_{cyc} is the number of clock cycles taken to execute the program and T_{clock} is the clock cycle period. We assign an energy cost to each instruction in the instruction set. The energy cost is formed by three contributions:

$$E_{instr} = E_{base} + E_{data} + E_{inter-instr} \tag{3-7}$$

where E_{base} represents the energy consumption of the functional units activated during the execution of an instruction, E_{data} quantifies the variation in base energy of a given instruction due to the different operands, addresses, and data, and $E_{inter-instr}$ takes into account inter-instruction effects. There are two types of inter-instruction effects. The first

type of inter-instruction effect comes from the observation that certain instructions significantly change the power consumption of the succeeding instructions. This is a consequence of the fact that the switching activity of a circuit is always a function of both the present inputs and the previous state of the circuit. The second inter-instruction effect is related to the fact that the amount of resources available in the system is finite, and hence there can be scenarios leading to stalls and cache misses.

The total energy consumed by the processor while running the software program is:

$$\begin{aligned} E_{cpu} = {} & T_{clock} \cdot V_{dd} \cdot \sum_{j=1}^{N} (I_{instr,j} \cdot N_{cyc,j}) + \\ & T_{clock} \cdot V_{dd} \cdot N_{miss,rd} \cdot N_{cyc,rd\_stall} \cdot I_{instr,nop} + \\ & T_{clock} \cdot V_{dd} \cdot N_{miss,wr} \cdot N_{cyc,wr\_stall} \cdot I_{instr,nop} + \\ & T_{clock} \cdot V_{dd} \cdot N_{miss,fetch} \cdot N_{cyc,fetch\_stall} \cdot I_{instr,nop} \end{aligned} \tag{3-8}$$

where V_{dd} is the voltage power supply, T_{clock} is the clock cycle period, $I_{instr,j}$ is the current drawn during the execution of instruction j, $N_{cyc,j}$ is the number of clock cycles that takes to execute instruction j, N is the total number of instructions in the program, $N_{miss,rd}$ is the number of read misses occurring in data cache, N_{cyc,rd_stall} is the number of cycles that the processor stalls when a read miss occurs in data cache, $I_{instr,nop}$ is the current drawn during the execution of a NOP instruction, $N_{miss,wr}$ is the number of write miss occurring in data cache, N_{cyc,wr_stall} is the number of cycles that the processor stalls when a write miss occurs in data cache, $N_{miss,fetch}$ is the number of fetch misses occurring in instruction cache, $N_{cyc,fetch\_stall}$ is the number of cycles that the processor stalls when a fetch miss occurs in instruction cache. The first term of the equation represents the energy consumption for the ideal case in which all instructions and data of the program are

retrieved from the caches, while the remaining three terms represent the energy penalty due to the time spent by the processor stalling because of a read miss in data cache, a write miss in data cache, or a fetch miss in instruction cache. The $E_{instr,j}$ depends not only on the present instruction j but also on the previously executed instruction k.

$$E_{instr,j} = T_{clock} \cdot V_{dd} \cdot \sum_{j=1}^{N} (I_{instr,j} \cdot N_{cyc,j}) = E(j,k) \tag{3-9}$$

3.4.2 Cache Power Dissipation

The estimation of a cache's energy consumption is based on the analytical models provided in [42],[43]. Accurate estimation requires that the cache simulator maintains activity statistics of several metrics, such as number of hits and misses, number of tag comparisons, word-line activity, and bit-line activity. Figure 3-5 depicts the architecture of a conventional M-way associative cache. An M-way associative cache consists of *M* static RAM banks (also called "ways" or "slots"). Each bank row (also called a line or a block frame) stores a contiguous group of 2^B memory words and an address tag *T* that uniquely identifies the line within the cache. The k-th row of all M banks is called the k-th set. The total number of sets (number of pairs line-tag) is $S=2^I$.

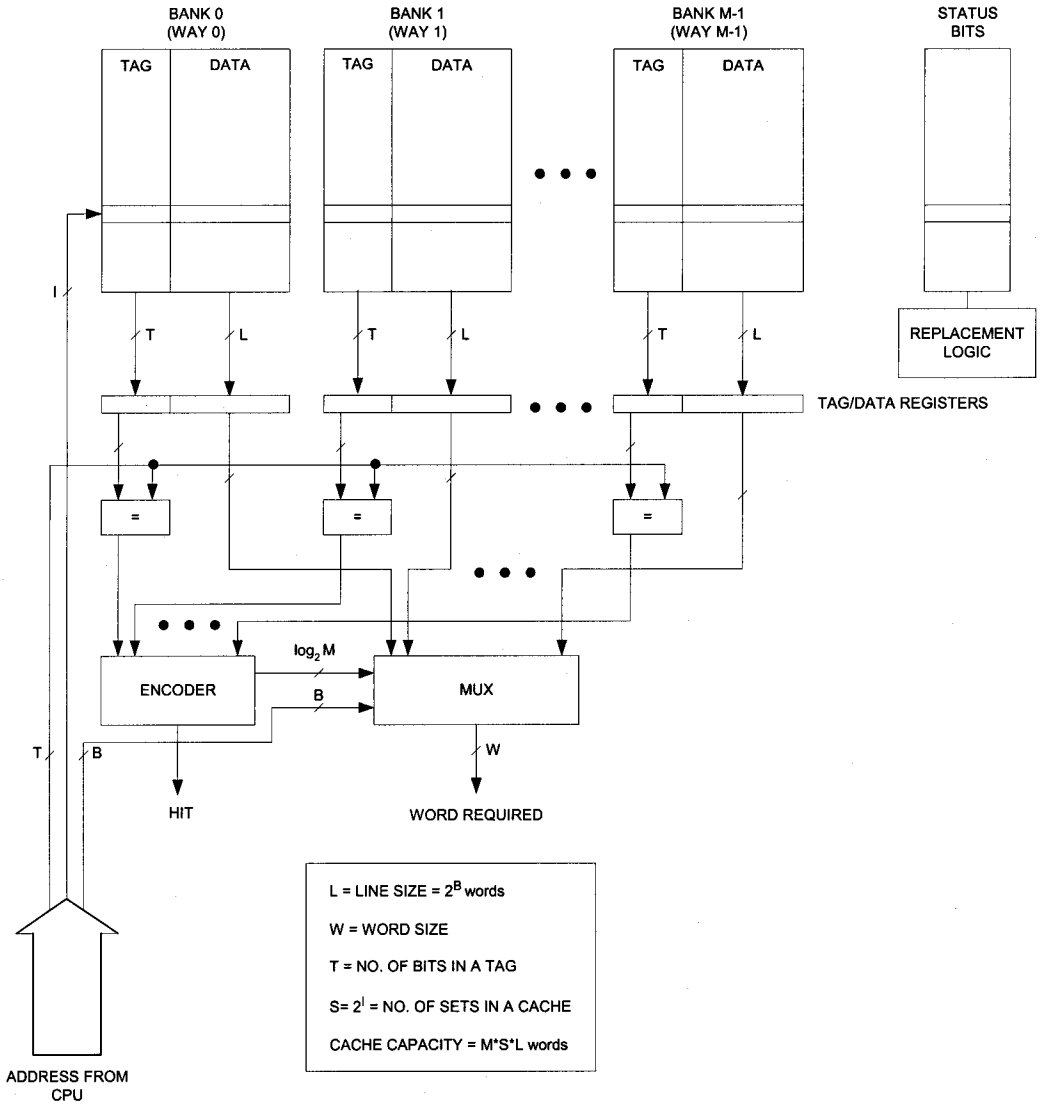

Figure 3-5 M-way associative cache

The address *VA=(T,I,B)* issued by the CPU can be thought of having two components: a block address *A* identifying the memory line (*A* is the CPU address shifted to the right by *B* bits, *A=VA>>B*), and a word address *B* identifying the word within the line.

A memory block with block address *A* can be placed into a cache in any one of the *M* block frames of set *A mod S*. The tag of the block is given by A *div S*. Figure 3-6 shows

the general architecture of a static RAM. The architecture is composed of three major blocks: memory cell array, decoders (row and column), and read/write circuitry (sense amplifiers and drivers). Figure 3-7 shows the internal structure of some of the SRAM key components.

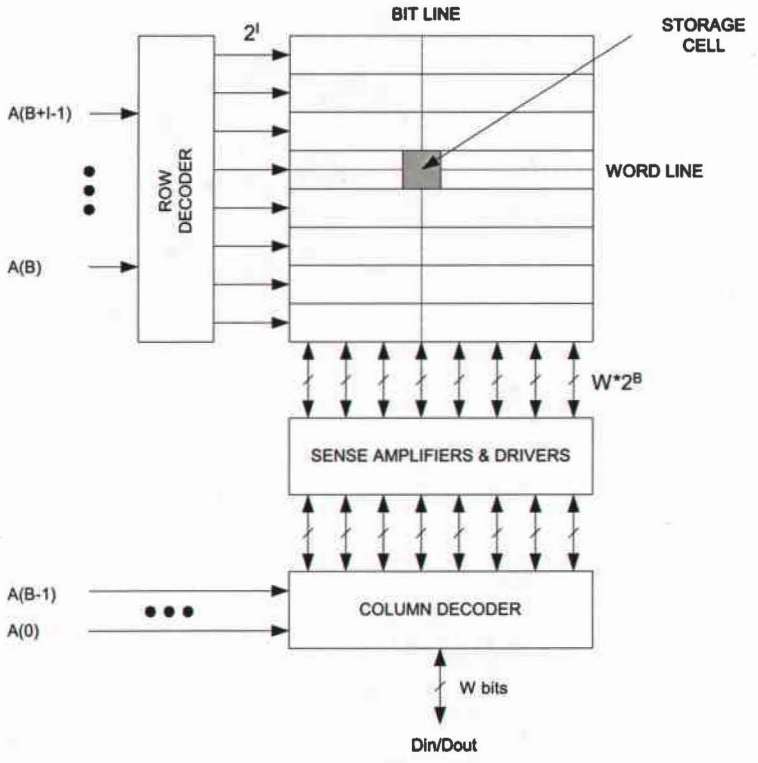

Figure 3-6 Simplified static RAM architecture

Since the capacitance switched during each cache access is directly related to cache size, line size, and associativity, the same relation holds for power consumption. The major components of energy consumption are in the bit-lines, word lines, output lines and input lines:

$$E_{cache} = E_{bitline} + E_{wordline} + E_{output} + E_{input} \qquad (3\text{-}10)$$

The energy dissipated in other cache components such us comparators, registers, data steering logic, control logic, and sense amplifiers is relatively small so their contribution can been neglected [45].

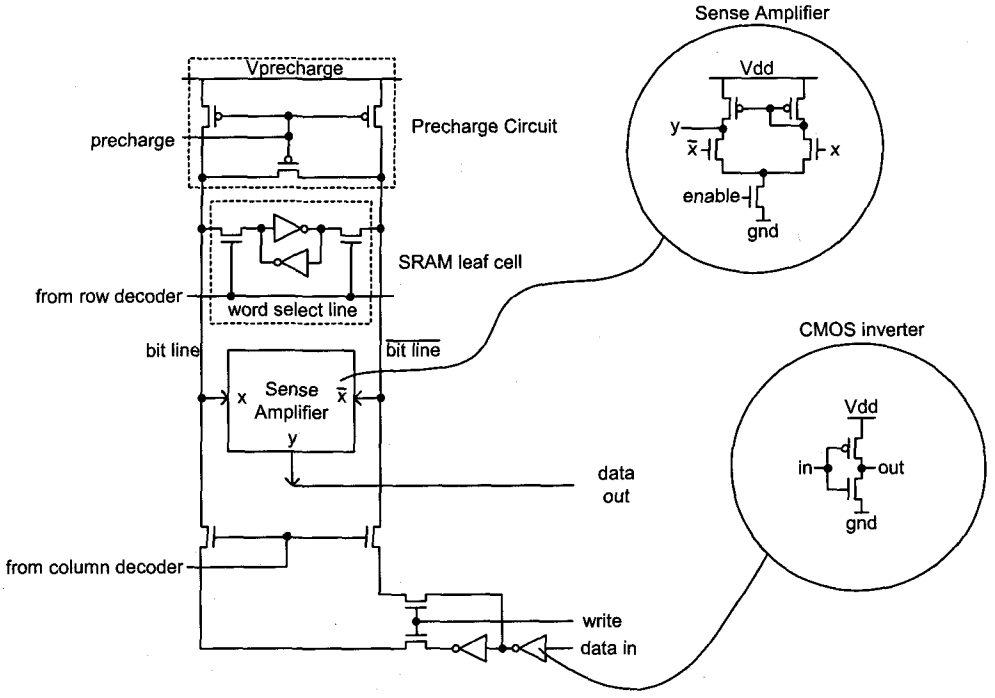

Figure 3-7 Static RAM internal structure

Given an M-way set associative cache, with a total data capacity of C bytes, a tag size of T bits, a line size of L bytes, and St status bits per block frame, the number of sets, S, can be expressed as:

$$S = \frac{C}{L \cdot M} \tag{3-11}$$

$$N_{rows} = S \tag{3-12}$$

$$N_{columns} = M \cdot (8L + T + St) \tag{3-13}$$

$$N_{bit\text{-}lines} = 2 \cdot N_{columns} \tag{3-14}$$

$$N_{cells} = N_{rows} \cdot N_{columns} \tag{3-15}$$

where N_{rows} the number of cell rows, $N_{columns}$ the number of cell columns, $N_{bit\text{-}lines}$ the number of bit lines, and N_{cells} the number of storage cells.

Bit line dissipation is due to charging/discharging of the bit lines during precharging and reads or writes of the bit cells. Every time there is a memory access the precharge circuit is activated. The capacitance switched in the precharge circuit itself is given by the gate capacitance of the 3 precharge transistors. The number of precharge circuits activated during a cache access is equal to $N_{columns}$.

$$C_{pre} = N_{columns} \cdot (2 \cdot C_{g-pre-tr-V} + C_{g-pre-tr-H}) \tag{3-16}$$

where $C_{g\text{-}pre\text{-}tr\text{-}V}$ is the gate capacitance of one vertical precharge transistor and $C_{g\text{-}pre\text{-}tr\text{-}H}$ is the gate capacitance of the horizontal precharge transistor.

The load capacitance seen by the precharge circuit to switch the bit lines of a cell is given by the series of the drain capacitance of the two pass transistors plus the capacitance of the bit line wire. Since there are N_{rows} cells in a column:

$$C_{bitline-pre} = N_{row} \cdot (0.5 \cdot C_{d-pass-tr} + C_{bitwire}) \tag{3-17}$$

After precharge follows the actual read or write. During pre-charge, both the bit lines are set to V_{dd}, and therefore, when the actual read or write cycle starts, or when the word line is asserted, one of the two bit lines must be discharged. Discharging a bit line requires discharging the drain capacitance of the vertical and horizontal precharge transistors attached to the bit line. The load seen by the column decoder is N_{rows} times the series of the drain capacitance of the two pass transistors and the capacitance of the bit line wire of

a cell. Therefore, the effective load capacitance switched during a read or a write is given by:

$$C_{bitline-r/w} = N_{rows} \cdot (0.5 \cdot C_{d-pass-tr} + C_{bitwire}) + C_{d-pre-tr-V} + C_{d-pre-tr-H} \quad (3\text{-}18)$$

and the energy dissipated in the bit lines is:

$$E_{bitline} = \frac{1}{2} \cdot V_{dd}^2 \cdot (C_{bitline-pre} + C_{bitline-r/w} + C_{pre}) \cdot N_{acc} \quad (3\text{-}19)$$

where N_{acc} is the total number of cache accesses.

Word-line dissipation is due to the large capacitive load that the row decoder has to drive whenever a row is read or written. The load capacitance seen by a row decoder driver is given by the gate capacitance of all pass transistors in the row plus the wire capacitance of the word selection line. Since, there are $N_{columns}$ cells in a row, each cell has 2 pass transistors, and during a memory access only one row driver at a time is active:

$$C_{wordline} = (2 \cdot C_{g\text{-}pass\text{-}tr} + C_{wordwire}) \cdot N_{columns} = (2 \cdot C_{g\text{-}pass\text{-}tr} + C_{wordwire}) \cdot M \cdot (8 \cdot L + T + St)$$
$$E_{worline} = 0.5 \cdot C_{wordline} \cdot V_{dd}^2 \cdot N_{acc} = (2 \cdot C_{g\text{-}pass\text{-}tr} + C_{wordwire}) \cdot M \cdot (8 \cdot L + T + St) \cdot V_{dd}^2 \cdot N_{acc}$$

(3-20)

where N_{acc} is the total number of cache accesses.

Output-lines dissipation occurs when the output drivers drive a new value out of the cache. The cache can drive values externally either toward the CPU or in case of a miss toward a lower level memory.

$$E_{output} = E_{out-addr} + E_{out-data}$$
$$E_{out-addr} = \frac{1}{2} \cdot V_{dd}^2 \cdot (N_{out-addr} \cdot C_{out-addr}) \quad (3\text{-}21)$$
$$E_{out-data} = \frac{1}{2} \cdot V_{dd}^2 \cdot (N_{out-data} \cdot C_{out-data})$$

where $C_{out\text{-}addr}$ is the input capacitance of the address lines, $C_{out\text{-}data}$ is the input capacitance of the data lines, $N_{out\text{-}addr}$ and $N_{out\text{-}data}$ are the total number of transitions on the address and data lines. The capacitive load can be quite different depending whether the destination is on-chip or off-chip.

Input lines dissipation accounts for the power consumption occurring in the decoder because of the input address transitions. The capacitance seen is given by:

$$C_{input} = C_{g-dec} \cdot S \cdot (2M+1) + C_{addrwire} \tag{3-22}$$

where $C_{g\text{-}dec}$ is the gate capacitance of the decoder and $C_{addrwire}$ is the wire capacitance of the common address wire feeding the decoder in each RAM.

Therefore the input energy dissipation is given by:

$$E_{input} = \frac{1}{2} \cdot V_{dd}^2 \cdot N_{in-addr} \cdot C_{input} \tag{3-23}$$

where $N_{in\text{-}addr}$ is the number of transitions in the address input lines.

3.4.3 Main Memory Power Dissipation

In order to analyze the energy consumption of main memory we used the analytical model described in ref. [46]. The main sources of power dissipation are: the memory cell array, the row decoder, the column decoder, and the periphery circuits. Given an m×n bit memory the power dissipated on each access is given by:

$$\begin{aligned} P_{mem} &= V_{dd} \cdot I_{mem-acc} \\ I_{mem-acc} &= m \cdot i_{act} + m(n-1) \cdot i_{hld} + (m+n) \cdot i_{dec} + i_{per} \end{aligned} \tag{3-24}$$

where $I_{mem\text{-}acc}$ is the current drawn during each memory access. Since on each memory access m cells are selected and m·(n-1) are not, $m \cdot i_{act}$ represents the current drawn by the

m selected cells, $m \cdot (n-1) \cdot i_{hld}$ the retention current of the $m \cdot (n-1)$ cells not selected, $m \cdot i_{dec}$ the current drawn by the row decoder, $n \cdot i_{dec}$ the current drawn by the column decoder and i_{per} the current drawn by the periphery circuits. At high clock frequency the power consumption contribution given by the active current is much bigger than all the other components.

$$I_{mem\text{-}acc} \approx m \cdot i_{act} \tag{3-25}$$

In case the main memory is a DRAM rather than an SRAM, the current drawn during a memory access is different for a single access, a first time access from a page in burst mode or a subsequent access from a page in burst mode. We assume that the current drawn in the case of a single access and in the case of a first time access from burst mode are the same.

$$I_{mem-acc} = m \cdot \frac{N_{first} \cdot i_{act-first} + (N_{acc} - N_{first}) \cdot i_{act-sub}}{N_{acc}} \tag{3-26}$$

where N_{first} is the number of single and first time accesses, N_{acc} is the total number of accesses, $i_{act\text{-}first}$ is the current drawn by an active cell in the case of single or first access from a page, and $i_{act\text{-}sub}$ is the current drawn by an active cell in the case of subsequent access from a page in burst mode.

3.4.4 Bus Power Dissipation

In sub-micron technologies bus power is a significant part of the total power. Execution time and bus power are inversely related. A smaller bus width implies less wire capacitance and hence less power, but requires more bus transfers and hence a higher execution time. The estimation of buses power dissipation is based on the key

observation that every memory and peripheral access implies a data transfer over a communication bus. The total number of cache accesses N_{acc} is a measure of traffic on the CPU-cache bus, the number of cache misses N_{miss} is a measure of traffic on the cache-main memory bus, the number of peripheral references N_{per} is a measure of traffic on the peripheral bus (bus between main memory and the peripheral devices). Given this traffic and assuming that on average at most half the bits will toggle, we can compute the switching activity on the bus. Given the bus switching activity and the bus capacitance, we can finally compute the power consumption.

$$P_{bus} = N_{bus} \cdot \frac{1}{2} \cdot C_{bus} \cdot V_{dd}^2 \cdot \alpha \cdot \frac{1}{T_{exec}} \tag{3-27}$$

where N_{bus} is the number of transactions on the bus, C_{bus} is the bus capacitance, α is the switching activity and T_{exec} is the time the application program runs on the system.

If we assume that there are $m=N_{bus}/T_{exec}$, n-bit items, transmitted per unit time, on a bus of width w:

$$\alpha = \left(\left\lceil \frac{n}{w} \right\rceil transfer/item \right) \cdot (w\ bit/transfer) \cdot (m\ item/s) \cdot \left(\frac{1}{2} transition/bit \right) \tag{3-28}$$

Since a bus transaction involves an address bus, a data bus and a few control signals, this equation is valid only for $n \leq w$. For $n > w$ we have to correct the equation to account for the fact that address and control lines will switch only on the first transaction of the requested transfer. However exploiting the locality of reference property, we can reasonably assume that the switching activity due to address and control lines can be neglected with respect to the switching activity due to the data lines. Under this assumption we can modify the switching activity equation as follows:

$$\alpha = \left(\left\lceil \frac{n_d}{w_d} \right\rceil transfer/item \right) \cdot (w_d \, bit/transfer) \cdot (m \, item/s) \cdot \left(\frac{1}{2} transition/bit \right) \quad (3\text{-}29)$$

where n_d is the number of data bits to be transferred and w_d is the width of the data bus. The bus capacitance is given by:

$$C_{bus} = B_{len} \cdot C_{wire} \quad (3\text{-}30)$$

where B_{len} is the length of the bus and C_{wire} the wire capacitance of the bus. Assuming a simple one-layer metal model the wire capacitance has only two components: the line-to ground capacitance and the line-to-line capacitance. No crossover capacitances come into play.

$$C_{wire} = N_{lines} \cdot C_{line} + (N_{lines} - 1) \cdot C_{inter-line} \quad (3\text{-}31)$$

where N_{lines} is the bus width (number of bit lines of the bus), C_{line} is the line to ground capacitance of a bit line, and $C_{inter\text{-}line}$ is the line to line capacitance of two adjacent lines. The line-to-ground capacitance and the line-to-line capacitance can be computed using the models provided in [67].

$$\frac{C_{line}}{\varepsilon} = \frac{W}{H} + 3.28 \cdot \left(\frac{T}{T+2H} \right)^{0.023} \cdot \left(\frac{S}{S+2H} \right)^{1.16} \quad (3\text{-}32)$$

$$\frac{C_{inter-line}}{\varepsilon} = 1.064 \cdot \left(\frac{T}{S} \right) \cdot \left(\frac{T+2H}{T+2H+0.5 \cdot S} \right)^{0.695} + \left(\frac{W}{W+0.8 \cdot S} \right)^{1.4148} \cdot \left(\frac{T+2H}{T+2H+0.5 \cdot S} \right)^{0.804} + + 0.831 \cdot \left(\frac{W}{W+0.8 \cdot S} \right)^{0.055} \cdot \left(\frac{2H}{2H+0.5 \cdot S} \right)^{3.542} \quad (3-33)$$

where ε is the dielectric permittivity, W is the width of the metal line, T is the thickness of the metal line, H is the thickness of the dielectric and S is the clear spacing between parallel lines. Those equations are valid for the following ranges:

$$0.3 \le W/H \le 10 \; and \; 0.3 \le S/H \le 10 \; and \; 0.3 \le T/H \le 10 \qquad (3\text{-}34)$$

3.4.5 Peripheral Power Dissipation

The technique used for estimating power consumption of peripheral cores is similar in idea to the one used for microprocessors. The key is to consider the peripheral core as executing a sequence of high-level instructions and being able to construct a power-per-instruction lookup table, so that a system level simulation executing the application program can be used to record the power consumption.

Generally, for a processor, the term instruction is used to mean an atomic action available for programming the desired behavior. However, for a peripheral the term is used in a more "relaxed" way. Here, the term instruction means an action that together with all other actions of the instruction set describes the peripheral's functionality [29]. The peripheral is modeled in terms of a set of instructions $\{i_1, i_2, i_3, \ldots, i_n\}$ and a set of power modes $\{m_1, m_2, m_3, \ldots, m_k\}$. The instruction set must be complete and not redundant. In other words, the instructions must be sufficient to cover the entire functionality F of the peripheral *C*, and there must be no two instructions covering the same function. Formally, this is equivalent to say that the two following properties must hold:

$$\textit{Completeness Property}: \; i_1 \cap i_2 \cap i_3 \cap \ldots \cap i_n = F(C) \qquad (3\text{-}35)$$

$$\textit{No redundancy Property}: \; i_j \cap i_k = \emptyset \text{ for } j,k = 1,2, \ldots, n \text{ and } j \neq k \qquad (3\text{-}36)$$

For each instruction we exploit the dependency of power consumption both on the instruction's input data and on the preceding instructions. An instruction's power dependency on input data can take three forms: 1) power is directly dependent on the

instruction's input data, 2) it is dependent on the density of 1's in the instruction's data, or 3) it is independent. Inter-instruction dependency accounts for the fact that certain instructions significantly change the power consumption of the following instructions. This effect is modeled by associating a set of power modes m_1, m_2, m_3, ..., m_k to each core. Formally we define a core's power model as a 6-uple:

$$< M, I, O, f_N, f_O, m_1 > \tag{3-37}$$

where M is a set of power modes $\{m_1, m_2, \ldots, m_k\}$, I is a set of inputs $\{i_1, i_2, \ldots, i_n\}$, O is the output (in our case power dissipation), f_N is a transition function that computes the next mode given current mode and input (f_N maps $M \times I \rightarrow M$), f_o is a function that computes the output ($f_o: M \times I \rightarrow O$), and m_1 is the initial mode.

In order to obtain power consumption we pursue a four-step procedure. First, we profile the application program for requests to and from the peripherals. The number and frequency of accesses to a peripheral is a measure of its switching activity. Second, we decompose the various types of tasks requested into instructions and map them into abstract functional units that are used to estimate complexity, i.e., gate count. Once we have switching activity and complexity, we can create the power per instruction lookup table. Third, we execute the peripheral model to generate the corresponding trace. Lastly, given the instruction trace and the power per instruction lookup table we compute power consumption. Figure 3-8 summarizes the four-step procedure proposed to estimate power consumption of peripherals.

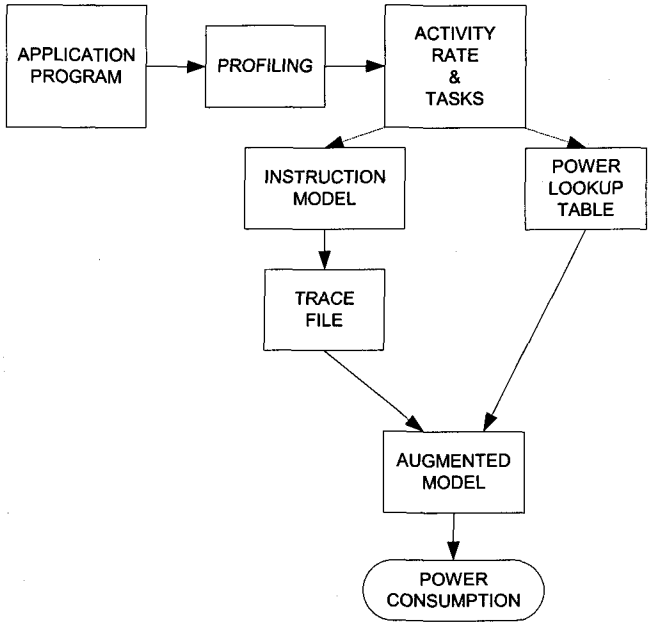

Figure 3-8 Peripheral power estimation procedure

Figure 3-9 shows how a generic core's model can be augmented in order to make power estimation possible.

```
// augmenting a functional model for power estimation
power_mode  = idle;
total_power = standby_power;

build_list(L);  // from parsing the trace file

while L is not empty {
  GetListItem(L);
  i = L.instruction;      // current instruction
  d = L.data;             // data passed into the current instruction
  t = L.dependencytype;   // instruction vs. data dependency type:
                          // independent, density_dependent, data_dependent.

  power_mode = NextPowerMode(power_mode, i);
  m = power_mode;

  if (t == independent) { // i is independent of its data
       total_power += PowerLookUpTable[m][i][0];
  }
  if (t == density_dependent) { // i is dependent on the density of its data
    total_power += PowerLookUpTable[m][i][1+GetDensity(d)];
  }
  if (t == data_dependent) { // i is dependent on its data) {
    total_power += PowerLookUpTable[m][i][2+GetIndex(d)];
  }
}
```

Figure 3-9 Augmenting the model of a core for power estimation

3.4.6 Interconnect Power Dissipation

In deep sub-micron technology, interconnect capacitance cannot be scaled down as much as gate capacitance. The relative increase of interconnect capacitance with respect to gate capacitance results in an increased importance of the role of interconnect for power estimation [2],[32]. There are cases reported in the literature, where the power due to interconnections is, excluding off chip driving, up to 46% of the total chip power [31]. The interconnection power consumption can be obtained as:

$$P_{interconnects} = \frac{1}{2} \cdot V_{dd}^2 \cdot C_{interconnects} \cdot A_{avg} \qquad (3\text{-}38)$$

where A_{avg} is the average switching activity of the system and $C_{interconnects}$ is the average interconnect capacitance. The average interconnects capacitance is directly related to the average interconnect length:

$$C_{interconnects} = R_{len} \cdot C_{line} \tag{3-39}$$

where C_{line} is the capacitance per unit of length of the routing wires, and R_{len} is the average interconnect length. C_{line} can be computed using the model proposed in [67], while R_{len} can be estimated using the well-known Rent's rule [32],[5],[30]. Rent's rule is stated as follow:

$$N_p = F \cdot (N_g)^r \tag{3-40}$$

where N_p is the total number of pins (input or outputs), N_g is the total number of gates, F is the average number of ports per gate, and r is an empirical constant known as the Rent exponent. F is a function of the specific gate library, however its value typically varies in a very limited range, so it is usually assumed to be 3.40. Given the average gate fan F, the gate count N_g and the pin count N_p we can derive r:

$$r = \frac{\log N_p - \log F}{\log N_g} \tag{3-41}$$

In order to compute the average interconnects length we use a closed form formula proposed in [32].

$$R_{len} = R(r) \cdot \frac{H(K,r,1)}{H(K,r,2)} \tag{3-42}$$

with:

$$\begin{aligned} H(K,r,i) &= \frac{2^{K(2r-i)} - 1}{2^{2r-i} - 1} \\ R(r) &= \frac{4 \cdot R_a(r) + 2 \cdot R_d(r)}{6} \end{aligned} \tag{3-43}$$

where:

$$R_a(r) = \frac{(r-1)}{(r+1)} \cdot \frac{3^{2r+2} - (r+4)\cdot 2^{2r+2} + (4r+7)}{3^{2r+1} - (2r+7)\cdot 2^{2r} + (4r+5)}$$
$$R_d(r) = \frac{(r-1)}{(r+1)} \cdot \frac{4^{2r+1} - 3^{2r+2} + 3\cdot 2^{2r+1} - 1}{4^{2r} - 3^{2r+1} + 3\cdot 2^{2r} - 1} \quad (3\text{-}44)$$

$$\text{and } K = \frac{1}{2} \cdot \log_2 N_g \quad (3\text{-}45)$$

3.4.7 Off-Chip Power Dissipation

A source of power dissipation often overlooked or ignored is the power used for off chip driving. The relative importance of its contribution is usually significant and sometimes tends to become dominant, especially with deep-submicron technologies. There have been cases reported in the literature where up to 70% of the total power consumption is due to off-chip driving [31]. Off chip driving power has two components. One component is the power used to drive the off chip capacitance. The other is the power consumed by the output driver itself. Typically off chip capacitance can range from 10 pF to 50 pF. Typically, the width decrease ratio of an output driving chain is 4, therefore the total driver chain capacitance is $C_{driver}=0.3\cdot C_{offchip}$. The off chip power consumption is:

$$P_{offchip} = \frac{1}{2} \cdot V_{dd}^2 \cdot f_{clock} \cdot \left(C_{offchip} + C_{driver}\right) \approx \frac{2}{3} \cdot V_{dd}^2 \cdot f_{clock} \cdot C_{offchip} \quad (3\text{-}46)$$

3.5 Summary

We have illustrated a method for power modeling suitable for an embedded system on a chip. The goal is to help designers to make more informed decisions from the earliest stages of system implementation. Collecting feedback on the impact of the

different choices as early as possible in the design flow, rather than just at the very end, shortens the design development time and reduces cost. Given a system and the application running on it, the method makes it possible to estimate the power consumption of each core in the system by exploiting an abstract notion of gate count and activity switching. Different from most of the work to date, we propose a high-level power estimation technique that operates entirely at the system level of abstraction. We do not use any gate-level (or transistor level) pre-characterization of the individual system components, to build the power per instruction lookup tables needed to evaluate power consumption. The reason is that, at the earliest stages of the design flow, a gate level implementation for all system components may not exist, or the provider may not want to disclose such intellectual property information. Other approaches' reliance on a gate level representation of all system components restricts the applicability of power evaluation only to architectures where components are already fully designed. In our approach lookup tables are simply built from target technology information and abstract functional decomposition of the system component behaviors.

The following chapters illustrate that the proposed technique has the potential for a significantly faster estimation of power consumption, albeit, at the expense of an acceptable loss of accuracy. The advantages are noticeable reduction in the computational complexity of problems such as design space exploration, synthesis optimization, and dynamic management of power aware systems.

CHAPTER 4
PEACE: A FRAMEWORK FOR POWER ESTIMATION AIMED AT THE CODESIGN OF EMBEDDED SYSTEM ON A CHIP

In this chapter, we provide a closer look at the implementation details of the power estimation framework illustrated in Figure 3-4. The current implementation supports only a subset of the conceptual framework proposed in Figure 3-4. In order to differentiate the conceptual framework from its implementation, we name the implementation PEACE (power estimation aimed at the codesign of embedded system).

The programming language used to implement PEACE is C++. Theoretically, any high level programming language, e.g., C or Java, would have been equally suitable. However, from a practical perspective, the choice of C++ is motivated by the fact that it is the underlying programming language on which SystemC is based [71]. SystemC is an open source development environment consisting of C++ class libraries and a simulation kernel for designing at the system and register transfer level. Besides providing a common high level language for modeling, analyzing and simulating both the hardware and software components of an embedded system, it can be also linked to commercial tools for hardware synthesis and simulation.

The embedded system components currently supported by PEACE are: processors, peripherals, and static memory. Figure 4-1 represents a simplified view of the framework implementation.

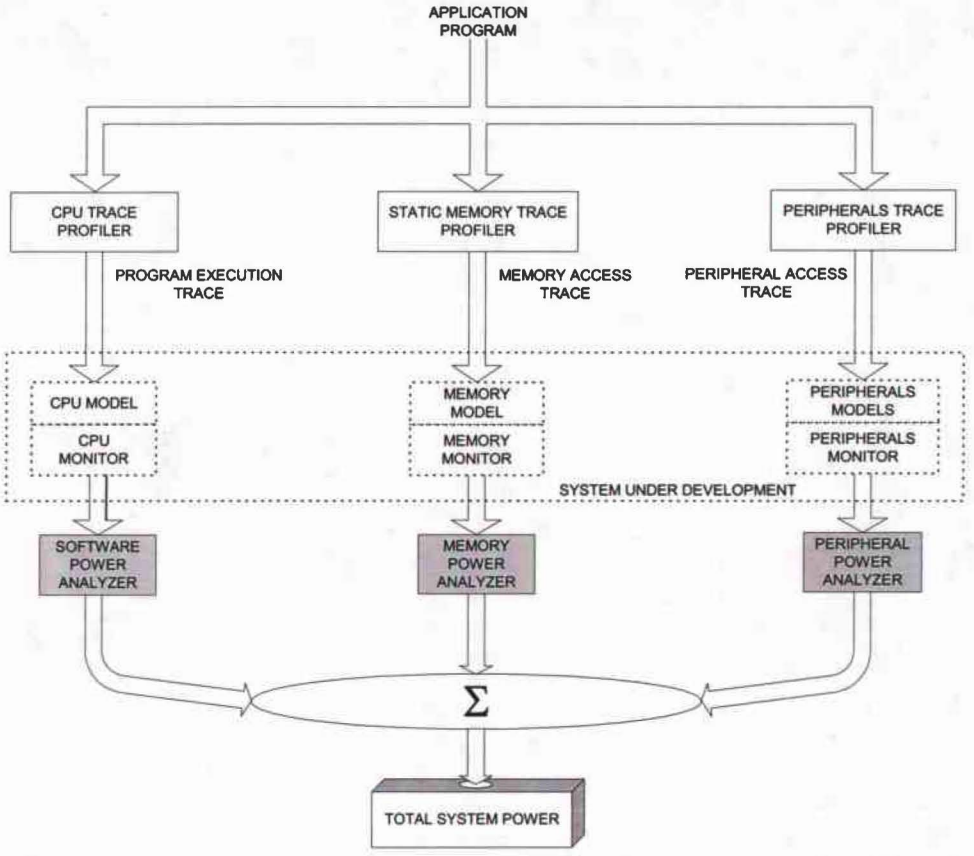

Figure 4-1 PEACE Framework Structure

A few preparatory steps are required before the application program can be effectively fed into PEACE. The application program is expected to be written in C/C++, compiled for the target processor, disassembled, and then parsed and distributed to the proper PEACE section. The role of the parser is to label the assembly instructions generated by the disassembler based on the component of the embedded system to which they are addressed. Once the application program has been successfully compiled, disassembled and parsed, it can be finally handed to one of the PEACE's profilers. The

profilers convert the instructions produced by the parser into an intermediate format more appropriate for the high level modeling of the embedded system. Figure 4-2 summarizes the steps needed before the application program can be submitted to PEACE.

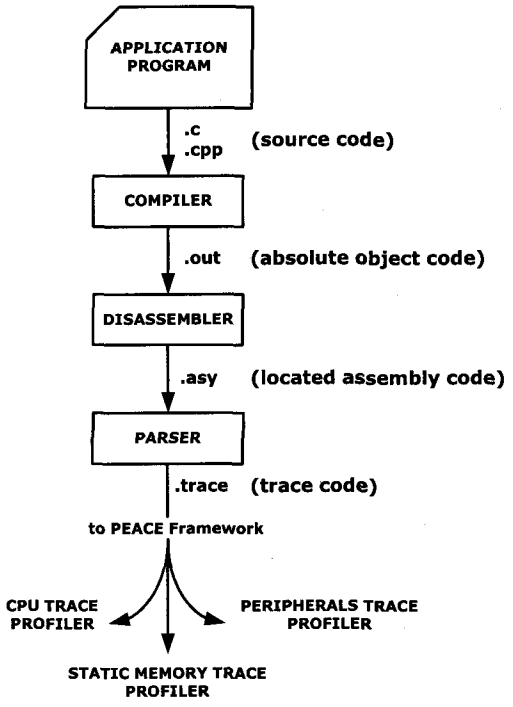

Figure 4-2 Pre-processing of the Application Program

As discussed before, the framework is formed by three distinct sections: the processor, the memory and the peripherals section. Although the specific implementation and the underlying equations used to estimate power consumption are different for each section, their structure is the same. Each section is formed by the following modules: a trace profiler, one or more models, a monitor for each model, and a power analyzer. The

"models" describe the system components' behavior, which in our case is done using the finite state machine paradigm. The models and monitors are associated either with different system components or different refinements of the same component. The "monitors" observe the execution of the associated models and characterize their power behavior. The "power analyzer" collects the information captured by the monitors and computes power consumption [72]. Figure 4-3 illustrates the structure and the different modules forming a generic PEACE's section.

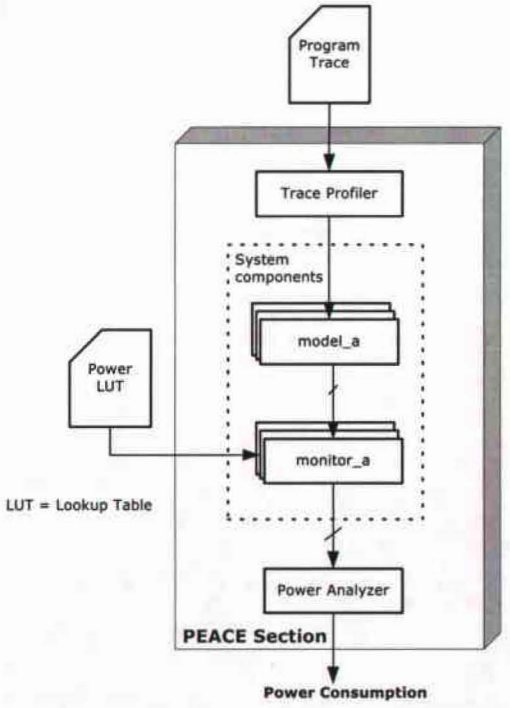

Figure 4-3 Structure and modules forming a generic PEACE's section

Figure 4-4 shows the detailed SystemC block diagram of a generic PEACE's section.

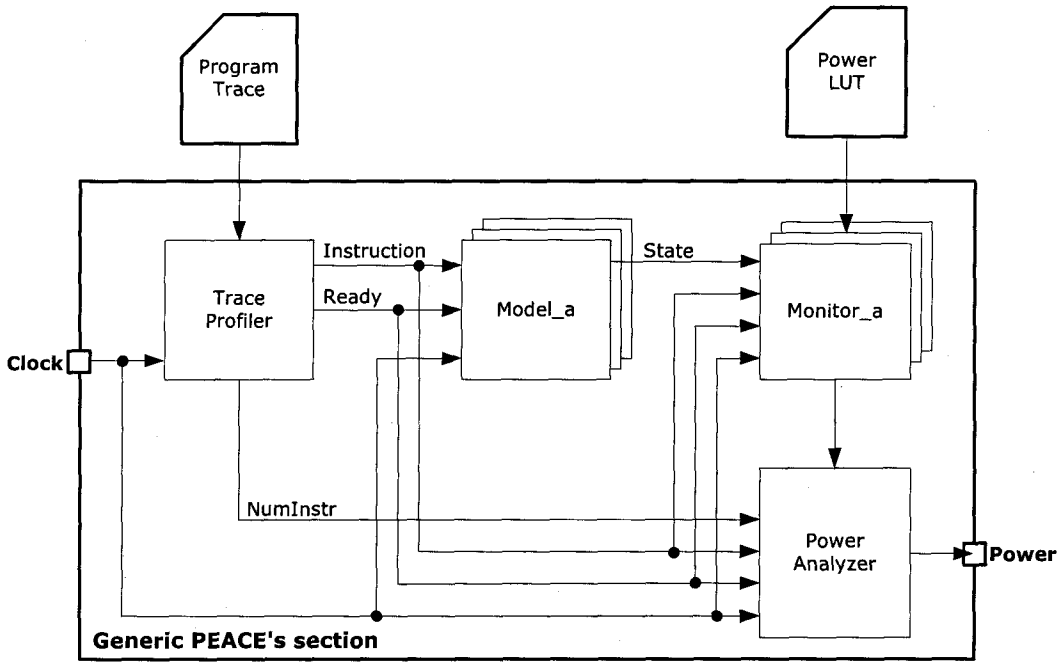

Figure 4-4 SystemC Design of a generic PEACE's section

The power look up table is generated by an external module called "design explorer". The "design explorer" analyzes the functional units forming the various components of the embedded system, estimates their complexity (i.e., total capacitance) based on the target technology chosen, and evaluates the power consumption of each functional unit. The information in the look up table is passed to the "monitors" of every PEACE section. The monitors use this information to characterize the power behavior of each system component. Figure 4-5 outlines the main steps involved in generating the power look up table.

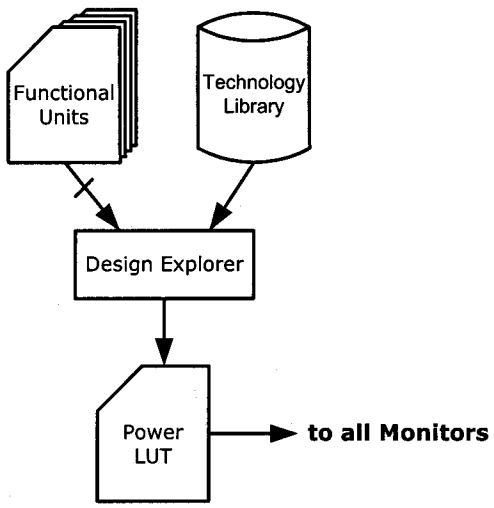

Figure 4-5 Generation of the Power Look up Table

Figure 4-6 illustrates the PEACE framework implementation and its interface with the application program, the target technology and the functional units

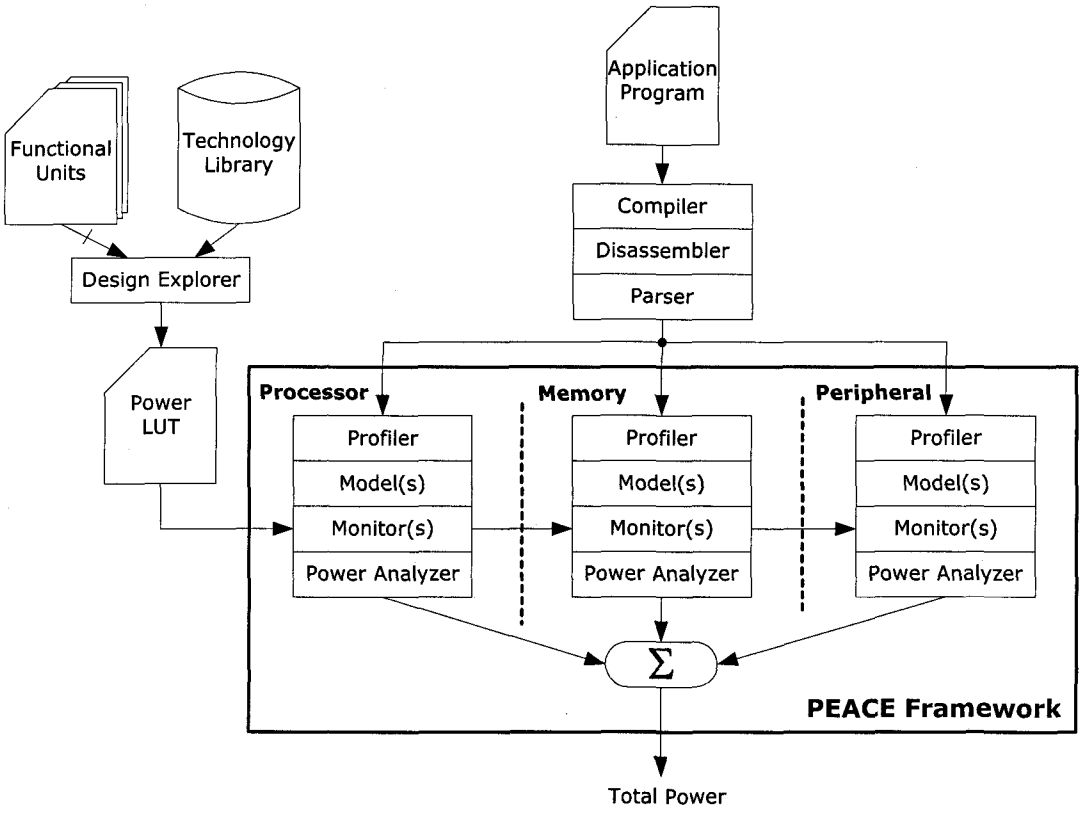

Figure 4-6 PEACE Input/Output Interfacing

CHAPTER 5
EXPERIMENTS AND RESULTS

In order to evaluate the accuracy and speed of our approach we apply it to different types of cores within a SOC. The cores considered are the baud generator of a universal asynchronous receiver/transmitter (UART), different sizes of embedded SRAMs, and the CPU of a microcontroller.

5.1 Peripherals Power Dissipation

Peripherals power estimation is based on the observation that their operation can be seen as the execution of a sequence of high level "instructions", hence power behavior can be can be captured in the form of power-per-instruction lookup tables. In order to produce the necessary power-per-instruction look up tables each peripheral is modeled in terms of a set of instructions and a set of power modes. Power modes take into account whether the current instruction can significantly change the power consumption of the following instructions. Power consumption is obtained by executing a system level simulation of the application program and the model associated to the peripheral [72].

In order to obtain power consumption we pursue a four-step procedure. First, we profile the application program for requests to and from the peripheral. The number and frequency of access to the peripheral is a measure of its switching activity. Second, we decompose the various requested tasks into instructions and map them into abstract functional units that are used to estimate complexity (gate count). Once we have switching activity and complexity, a power-per-instruction lookup table is created. Third, we execute the peripheral model to generate the corresponding trace. Lastly, given the

instructions trace and the power-per-instruction lookup table, we compute the power consumption.

5.1.1 Peripheral illustrative example

In order to illustrate and validate the suggested approach, we modeled and simulated the baud generator unit that clocks the universal asynchronous receiver/transmitter (UART) inside the Infineon's XC161CJ micro controller. We compared our system level approach both to the gate level approach and to direct measurements taken on the real hardware. The experimental results indicate that the system level approach is within 20% of the gate level estimation and within 12% of actual physical measurements. The system level approach has also the advantage of executing 3 orders of magnitude faster than the gate level approach.

The baud generator introduced consists of three functional units: a prescaler containing a selectable fractional divider and two fixed integer dividers, a 13-bit timer, and an output stage providing the outgoing baud rate. Figure 5–1 shows the baud generator architecture.

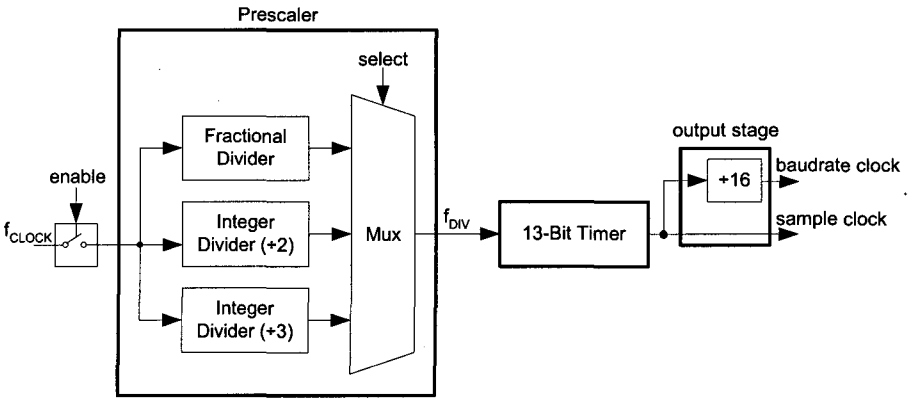

Figure 5–1 Baud generator architecture

The power behavior is described through the use of a finite state machine, where each state represents a power mode and the transition from state to state depends on the instructions created from the application program. The baud generator power model is illustrated in Figure 5–2.

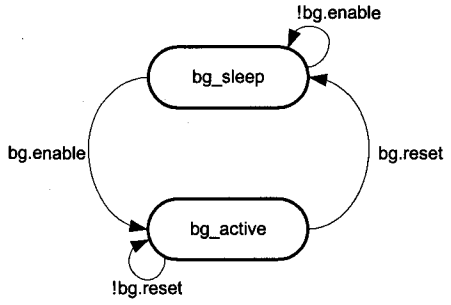

Figure 5–2 Baud generator power model

The total energy consumed by the baud generator during the execution of the application program is given by:

$$E_{bg} = T_{clock} \cdot \sum_{j=1}^{N} (n_{cyc,j} \cdot p_{inst,j}) = \sum_{j=1}^{N} (T_{clock} \cdot n_{cyc,j} \cdot p_{inst,j}) \quad (5\text{-}1)$$

where T_{clock} is the clock cycle period, $p_{instr,j}$ is the power dissipated during the execution of instruction j, $n_{cyc,j}$ is the number of clock cycles taken to execute instruction j and T_{exec} is the total execution time of the application program. The power per instruction can be computed as follows:

$$p_{instr,j} = \frac{1}{2} \cdot V_{dd}^2 \cdot \sum_{k=1}^{NF} C_k \cdot \alpha_{k,j} \quad (5\text{-}2)$$

where V_{dd} is the power supply voltage, NF is the number of functional units composing the baud generator, C_k is the total capacitance of functional unit k, and $\alpha_{k,j}$ is the switching activity occurring within the functional unit k in order to execute instruction j.

5.1.2 Setup and Results

To assess the validity of our approach, we have designed an experimental set up that a) allows us to collect system level power estimations in a dynamic execution scenario, b) compare the system level approach to the gate level approach, and c) compare the system level estimations to measurements on the actual hardware.

First, we implemented in C++ the system level model of the baud generator and the peripherals module of the PEACE Framework shown in Figure 4-1. The model of the baud generator includes only the power behavior. In order to achieve a good accuracy we took into account the dependency of each instruction on the previous ones. The

dependency is expressed through the use of a state machine. The transitions from one state to the other are triggered by the sequence of instructions.

To compute the power per cycle of each core's instruction, we employed Infineon's 0.25 μm CMOS technology. We have estimated the average capacitance of combinational cells and sequential cells separately. The estimations were obtained averaging the intrinsic capacitance of the cells in the target technology. The resulting values are stored in a look up table to make their access easier during the execution of the model. Figure 5–3 provides a closer look at the implementation details of our approach and compares it to the gate level approach. The "system level" side of Figure 5–3 shows how the peripheral's section of the PEACE framework is implemented. The "Design Explorer" analyzes the functional units forming the various components, estimates their complexity (i.e., total capacitance) based of the target technology chosen, and evaluates the power consumption of each functional unit. The "Application Profiler" parses the application program's extracts the instructions that affect the peripherals, and distribute them to the peripherals exercised. Different models and monitors are associated with either different peripherals or different refinements of the same peripheral. The monitors observe the execution of the associated models and characterize their power behavior. The "Power Analyzer" collects the information captured by the monitors and compute power consumption.

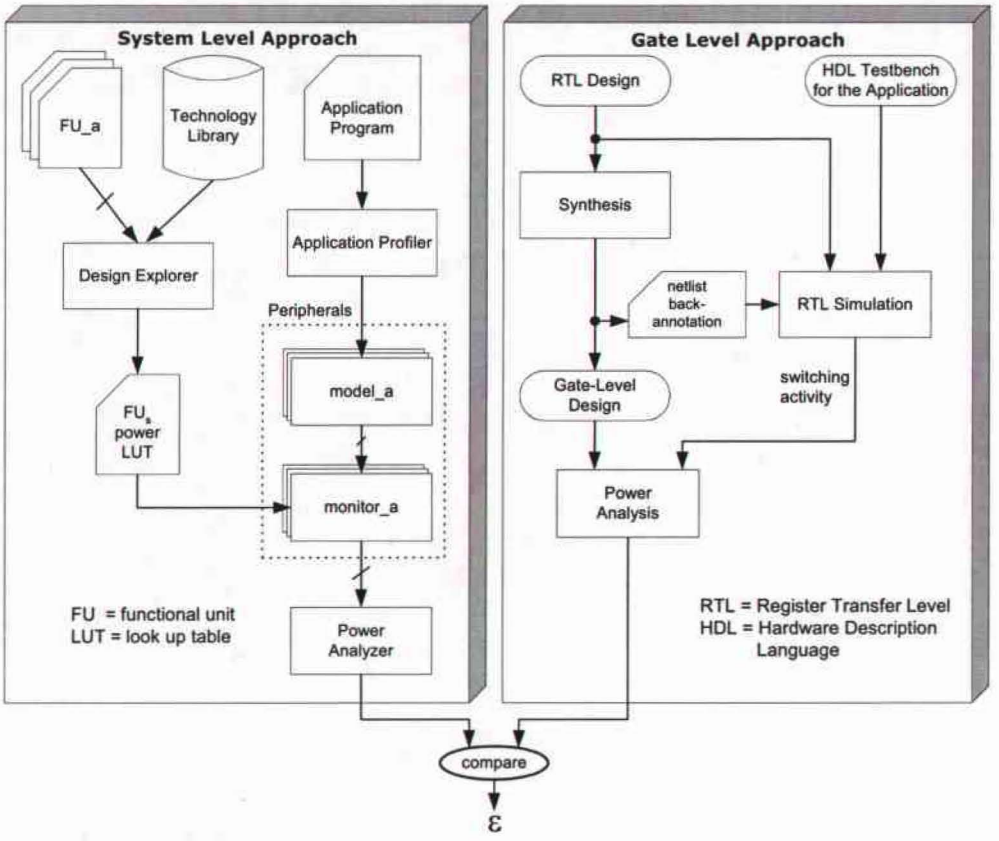

Figure 5–3 System level approach versus gate level approach

The experiments performed consisted, in running for 2000 clock cycles, twenty automatically generated application programs. In order to make comparative analysis possible a VHDL model of the baud generator has been used as reference. The VHDL model has been implemented at the Register Transfer Level (RTL) and then synthesized down to gate level using Synopsys synthesis tools [73]. Gate level power estimation has been performed using Synopsys Power estimation tools [74],[75]. In order to perform the

gate level power estimation, we have generated a set of VHDL testbenches replicating the application programs. Figure 5–4 summarizes the power per cycle dissipated by each instruction using gate level power estimation and our system level approach. The average difference is 9.71%.

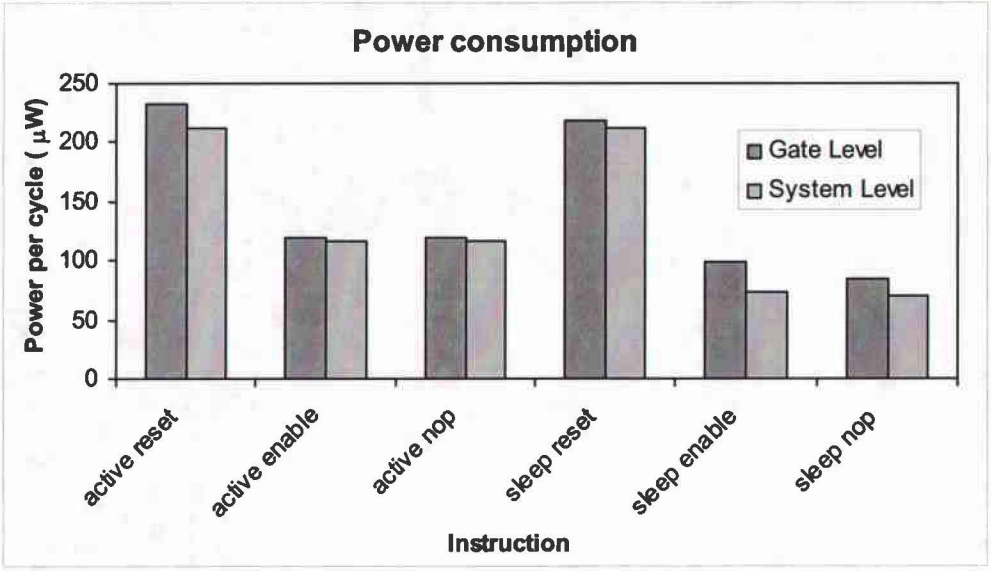

Figure 5–4 Power per cycle dissipated by each instruction

As can be noticed, the system level estimation is consistently lower than the gate level estimation. This is due to the fact that the level of details considered at the system level estimation is always lower, than the one used at the gate level. Power consumption under-estimation represent a serious issue in cases where the focus is worst case design analysis, rather than design tradeoffs exploration. Figure 5–5 show the energy consumed by the system during the execution of one of the twenty application programs benchmarked. The energy plots of all application programs benchmarked are reported in

Appendix A. The capability of profiling energy is particularly useful to gain insight into the system hot spots.

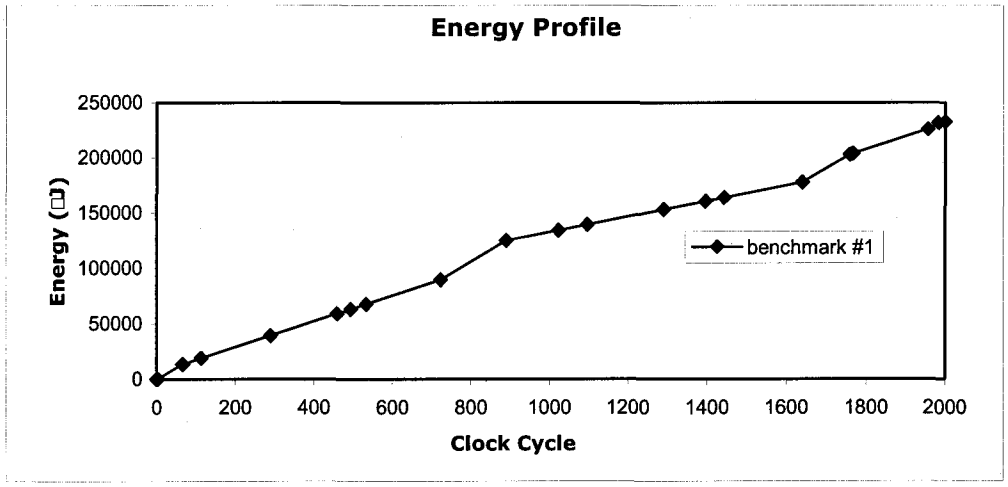

Figure 5–5 Plot of energy per clock cycles elapsed for benchmark #1

Figure 5–6 shows the scatter plot of the power dissipated by the benchmarks over an execution time of 2000 cycles. The average error is below 20%.

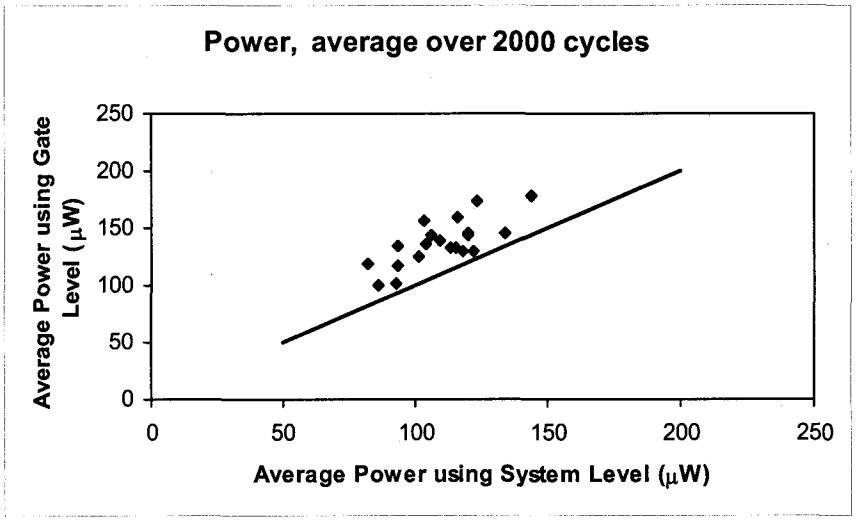

Figure 5–6 Scatter plot of average power for twenty benchmarks

Each point on the scatter plot depicts the average power of a benchmark. The abscissa represents the value of the average power obtained using the system level approach. The ordinate represents the value of average power obtained using the gate level approach. Ideally if there were no difference between the two methods all dots should lie on the solid line. Compared to gate level power estimation, our approach gives a speedup of 1343. Figure 5–7 summarizes the estimated current drawn by the benchmarks for both the system level approach, and the gate level approach and compares the estimations to the values measured on the real hardware.

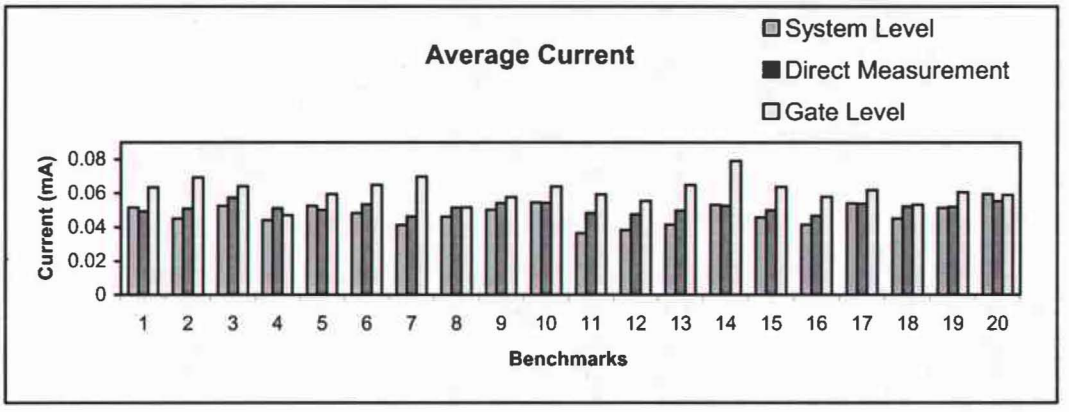

Figure 5–7 Average current consumption using a) system level approach, b) gate level approach, and c) direct measurement

The results indicate that on average the system level approach is 8.57% off the actual physical measurements. The trend shows that system level approach is on average closer to the actual measurements than gate level approach. This can be explained by the fact that the power models associated to gate level primitives are characterization based so they suffer from considerable bias when used in different conditions from the one used for the characterization. For the few points where the error between system level

estimation and actual measurement tend to be higher we noticed that the benchmarks contained a particularly high concentration of instructions for which the system level modeling is particularly difficult. Typical examples are instructions that involve reset or instructions that depend on the characteristic of the data processed. Figure 5–8 shows the scatter plot of the average current between system level approach and actual measurement.

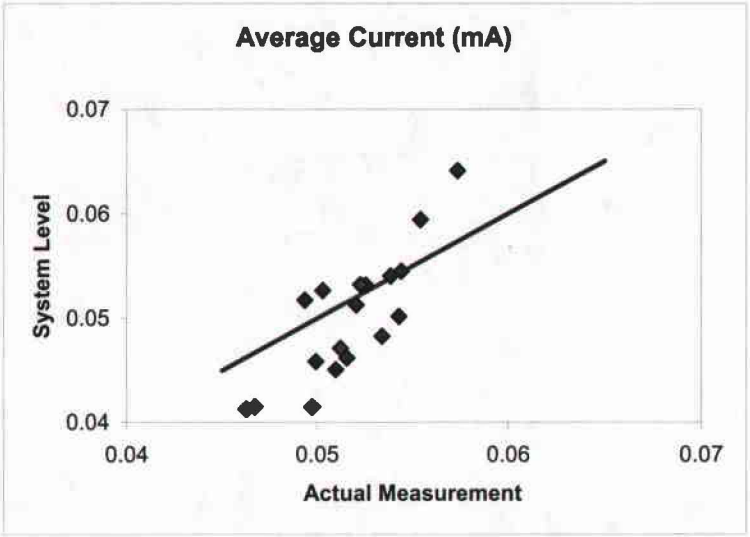

Figure 5–8 Scatter plot of average current

The accuracy is satisfactory to make power related system trade offs.

5.2 Memory Power Dissipation

Many of today's electronic products perform tasks that deal with very large quantities of data. As a consequence these systems require a significant amount of memory. Performance considerations, i.e., the intent to achieve minimum access time, and the aim to reduce i/o pins dictate designers to place as much memory as possible directly on-chip. As

a result, in many of today's applications, memory has become the primary power-consumption component of the chip [45]. High performance on-chip memories are typically static random access memories (SRAMs).

To estimate power consumption, we identify SRAM's main dissipating components and devise a model accordingly. The main sources of power dissipation are: the memory array, the row decoder, the column decoder, and the read/write circuits (sense amplifiers and drivers). A detailed description of the operation of all the components of a static SRAM can be found in ref. [75]. In order to compute the energy consumption we use the analytical models described in references [46] and [47]. The actual values of the SRAM internal capacitances are based on Infineon's 0.13 micron technology.

5.2.1 SRAM illustrative example

We illustrate and validate our methodology by applying it to four different SRAMs. The results obtained with our approach are compared with the results obtained applying SPICE. The technique is sufficiently accurate for making power related system level tradeoffs and it executes orders of magnitude faster than circuit level simulation. As a consequence, we can explore many more architectural tradeoffs before committing to a specific physical implementation. The power behavior of the SRAM is described through the use of a finite state machine, where each state represents a power mode and the transitions from state to state depends on the operations requested on the memory. A pictorial representation of the state machine is given in Figure 5–9.

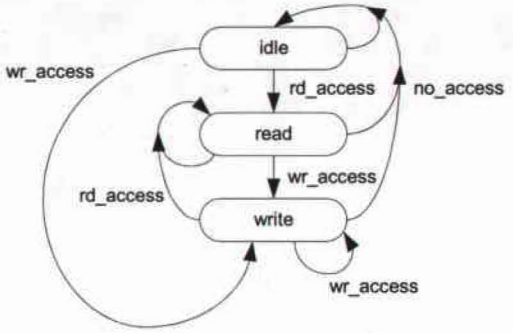

Figure 5–9 Power model of the static RAM

5.2.2 Setup and Results

We applied our modeling technique to four different sizes of SRAM: 1Kx4 bits, 256x16 bits, 8Kx8 bits and 4Kx16 bits. Figure 5–10 shows the layout of one of the SRAM devices used for validating the model.

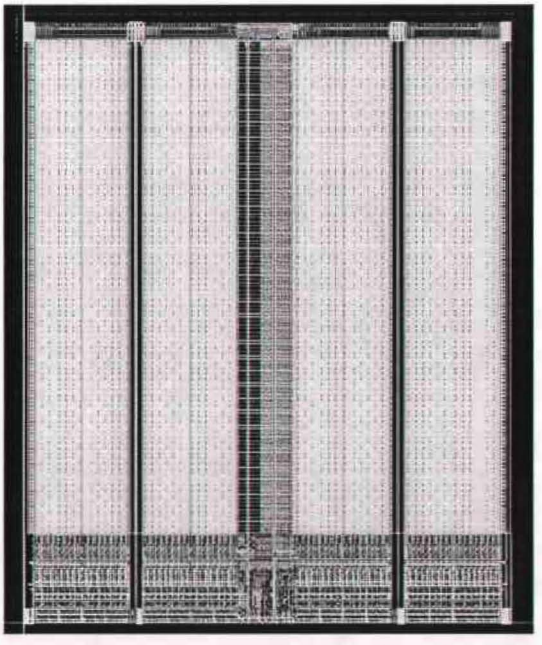

Figure 5–10 Layout of one of the four SRAM devices used to validate the model (4Kx16 bits)

Figure 5–11 summarizes the power dissipated when accessing the memories according to both SPICE and our system level approach. The average difference is about 10%, which is a satisfactory accuracy for making power related decisions at the system level [77],[78]. The experiments consisted of running 25 clock cycles of automatically generated memory accesses trace file. Generating the trace file automatically rather than obtaining it from a real application program implies that we do not exploit the locality of reference property. As a consequence the power profiles obtained are over pessimistic. The capability of profiling power consumption permits to gain insight into the system hot spots, so it can be used as an effective tool to identify opportunities for software optimizations.

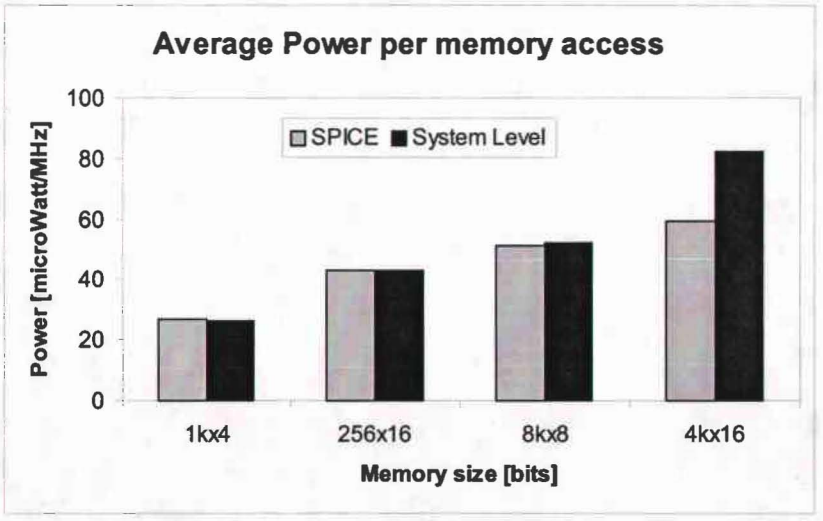

Figure 5–11 Average power dissipated per memory access

Figure 5–12 shows the energy consumed by one of the four SRAM when stimulated by the memory accesses trace file used as benchmark. The average error is below 9%. The speedup obtained using the system level approach is 7214.

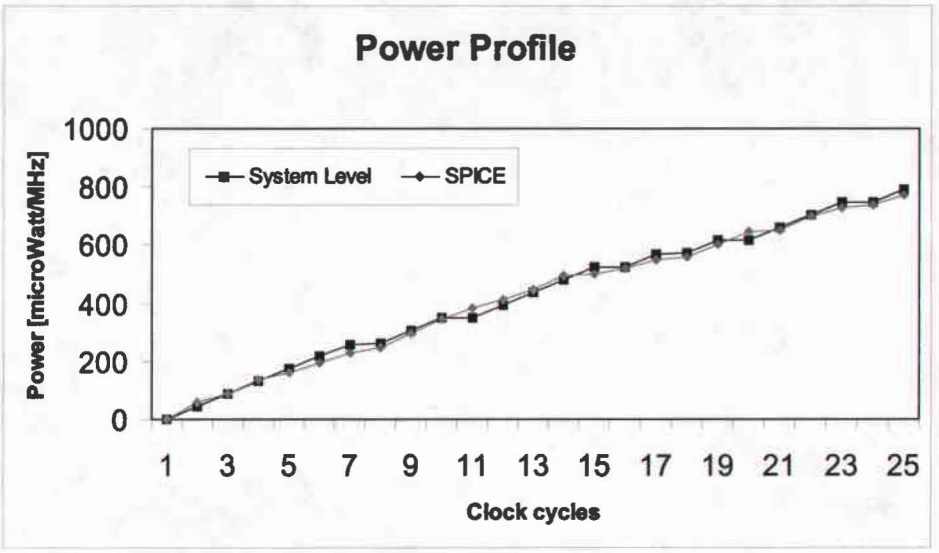

Figure 5–12 Power Profile (8Kx8 bit static RAM)

5.3 Processor Power Dissipation

In order to estimate the power consumption due to the execution of the application software on a processor, we rely on an Instruction Set Simulator (ISS). The ISS maintains detailed statistics of the processor's internal activity e.g., fetches, stalls, instruction execution frequency, internal register accesses, etc. Such statistics can be post-processed to compute power consumption. The processor is modeled in terms of a set of instructions and a set of power modes.

5.3.1 Processor illustrative example

In order to illustrate and validate our approach for processors we modeled and simulated the CPU inside the Infineon's XC161CJ micro controller. We compared our approach to the actual values measured on the hardware. The experimental results indicate that the system level approach is within 10 % of actual physical measurements.

A simplified view of the XC161CJ architecture is shown in Figure 5–13. Here a C166v2 processor is connected to program memory (PSRAM) and data memory (DSRAM). The connection to PSRAM is via the processor internal bus and a program management unit (PMU). The connection to data memory (DSRAM) is via the processor internal bus and a data management unit (DMU). The PMU handles all code fetches, while the DMU handles all data transfers. A high-speed system bus connects the DMU and the PMU. The data management unit acts also as a bridge between the high-speed system bus and the peripheral bus. Several peripherals are connected on the peripheral bus: two Universal Synchronous/Asynchronous receiver/transmitter (USART), two high-speed synchronous channels using the System Packet Interface protocol (SPI), an analog/digital converter (ADC), a general-purpose timer (GPT), a serial data link module (SDLM), two capture/compare units (CC) and an Inter-Integrated Circuit interface (IIC).

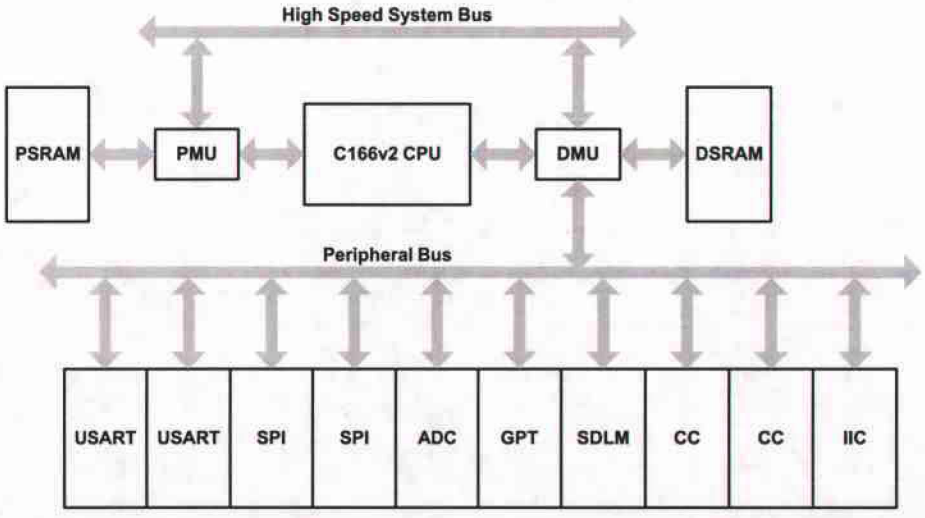

Figure 5–13 Simplified view of the XC161CJ architecture

In order to model power consumption only a small number of details are required. The CPU power model is illustrated in Figure 5–14.

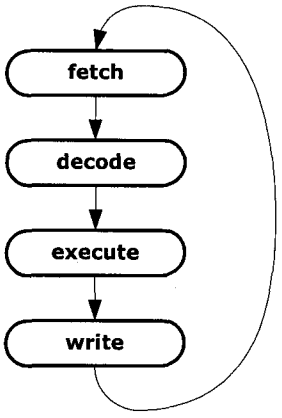

Figure 5–14 CPU power model

5.3.2 CPU Power Characterization

In order to model the CPU we used the instruction-based approaches given in [21],[29]. The idea behind instruction-based approaches is that "by measuring the current drawn by the processor as it repeatedly executes certain instructions, it is possible to obtain most of the information that is needed to evaluate the power cost of a program for that processor" [21]. The average power consumed by a processor while running a program is given by: $P_{avg} = I_{avg} \cdot V_{cc}$ where P_{avg} is the average power, I_{avg} is the average current, and V_{cc} is the supply voltage.

The power consumed by each instruction has been determined by constructing a loop with several instances of the same instruction. The current being drawn is then measured. For each instruction we have repeated the experiment at different times, and averaged the observed current. It is important to notice that while the values reported are specific to the

Infineon's CPU, the methodology used in developing the model is general. The current has been measured using a standard digital ammeter. The average current for a representative subset of the CPU instruction set is summarized in appendix A.

5.3.3 Setup and Results

In order to validate our approach and evaluate its accuracy we applied to the CPU fifteen different application programs. The application programs used to verify the system are: blinking_led.c, fibonacci.c, sieve.c, eigth_queens.c, pascal.c, bubblesort.c, insertionsort.c, quicksort.c, binarysearch.c, differentiation.c, matrix_arithmetic.c, terrain_navigation.c, stringsearch.c (brute force algorithm), euclid.c, and palindrome.c. As expected, a more refined model allows an increase in accuracy. The results obtained with our approach are within the 10% of actual physical measurements, and the average error is 6.29%. For each application program, Table 5-1 summarizes the current measured, the current estimated and the relative error.

Application Program	Average current measured (mA)	Average current estimated (mA)	Percentage Error
Blinking LED	11.01195	11.0657	0.49%
Insertion sort	11.99072	10.9888	8.36%
Bubble sort	11.8915	11.0105	7.41%
Quick sort	11.87383	11.0555	6.89%
Binary search	11.92426	10.9789	7.93%
Fibonacci's Series	11.99268	11.0206	8.11%
Eratosthenes' Sieve	11.98805	10.9511	8.65%
8 Queens Puzzle	12.05746	10.9751	8.98%
Pascal's Triangle	12.00589	10.9717	8.61%
Differentiation	11.52476	11.0307	4.29%
Terrain Navigation	12.14081	11.0333	9.12%
Matrix Arithmetic	11.83084	11.0447	6.64%
Euclid Algorithm (GCD)	11.44642	10.9637	4.22%
String Search	11.80786	10.9503	7.26%
Palindrome	11.00262	10.9503	0.48%

Table 5-1 CPU's current: measured vs. estimated

Figure 5–15 illustrates in a graphical form the values listed in Table 5-1.

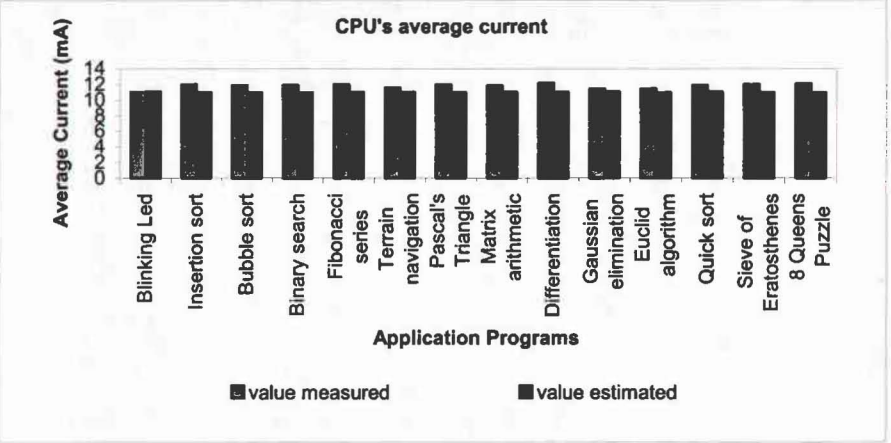

Figure 5–15 Current drawn by the application programs

5.4 Summary

In this chapter, we have illustrated and validated our system level methodology by applying it to different illustrative examples. We have shown that the methodology proposed is general: it can be used in the earliest stages of the design cycle, but also as design progresses. At different stages of the design cycle, the system description can be refined with increasingly accurate and detailed information. The results obtained from the experiments performed confirm that the technique is sufficiently accurate for making power related system level trade offs.

First, we modeled the baud generator inside the Infineon's XC161CJ microcontroller, and compared our system level approach with the gate level power estimations provided by a commercial CAD tool and with the actual measurements performed on the real hardware. The results were within 20% of the gate level estimation [72], and within 12% of the actual measurements. The execution time of our system level approach was three

orders of magnitude faster than the gate level approach. Second, we applied our approach to the existing design of four embedded SRAMs. The results were within 10% of circuit level power estimations obtained using SPICE [77],[78]. Last, we modeled the CPU inside the Infineon's XC161CJ micro-controller. In this case the model was based on more accurate and detailed information than the previous examples. The model built relies on the pre-characterization of the current drawn by each instruction of the CPU. The results obtained with our approach are within the 10% of the actual measurements and the average error is 6.29%.

In the next chapter we introduce opportunities for future extensions of our power estimation framework. In order to demonstrate the viability and the practical impact of the ideas proposed we introduce as a case study the high level design of an Aircraft Pressurization System (APS) to be mapped on a generic embedded platform consisting of two ASICs and a CPU.

CHAPTER 6
SYSTEM LEVEL DESIGN OPTIMIZATION

In this chapter we describe what future work is needed to broaden our power estimation framework to make it possible to assess multiple performance metrics and to accomplish stepwise modifications of the design. Modifying the design requires the capability of iterating through the different steps of the design flow so that various design tradeoffs can be generated and analyzed. There are many choices that designers can make toward the goal of improving the overall quality of a design. These choices regard different hardware/software partitioning, different types of architectural components, and different types of communication protocols. Figure 6-1 shows how the framework can be extended to foster multiple performance metrics assessment and iterative design modifications.

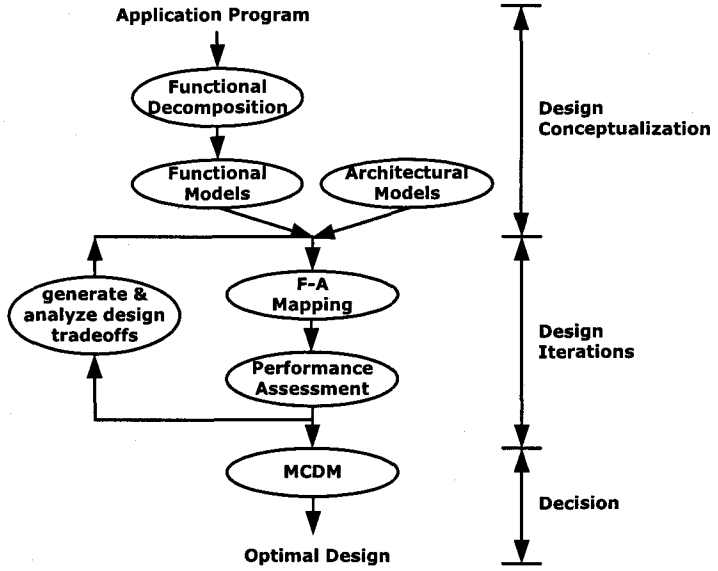

Figure 6-1 Extended Framework

The extended framework follows a 7 step scheme: 1) the application program is decomposed into primitive functional units, 2) a high level model of the functionality is built, 3) many different architectures are made available to implement the desired functionality, 4) the functionality is mapped into a specific architecture, 5) the performance metrics for the mapping under investigation are assessed, 6) step 4 and step 5 are repeated until new functionality-architecture mappings (in other words new design instances) can be generated, and 7) multi criteria decision making (MCDM) method is used to rank the various design instances and select the optimal one.

The problem of designing an embedded system is seen as the task of choosing from all possible design instances, the one that optimally satisfies functionality and performance constraints. In essence given a set of design alternatives and a set of decision criteria, the goal is to search for the best alternative (optimal solution). One of the fastest growing areas addressing this problem is the class of methods referred as Multi-Criteria Decision Making (MCDM) [79].

In this chapter we show how to analyze the tradeoffs resulting from the partitioning of the embedded system functionality into different architectures and we describe the use of MCDM methods to rank various design alternatives. We demonstrate the approach by applying it to the design of an Aircraft Pressurization System (APS) example and discuss the results obtained.

6.1 Motivation

The framework presented in the previous chapters is aimed to the assessment of power consumption, however because of its underlying structure it can be generalized to

any performance metric. Moreover, it can be extended to consider the effect of many performance metrics. If simultaneous performance metrics are considered, since some of them may conflict, the problem of identifying the best design alternative among many becomes extremely complex. As a consequence the performance assessment framework must be integrated with a decision process "engine" that after exploring all given design instances select the one that optimally satisfy all performance metrics involved. The integration between the decision process engine and the performance assessment framework is summarized in Figure 6-2.

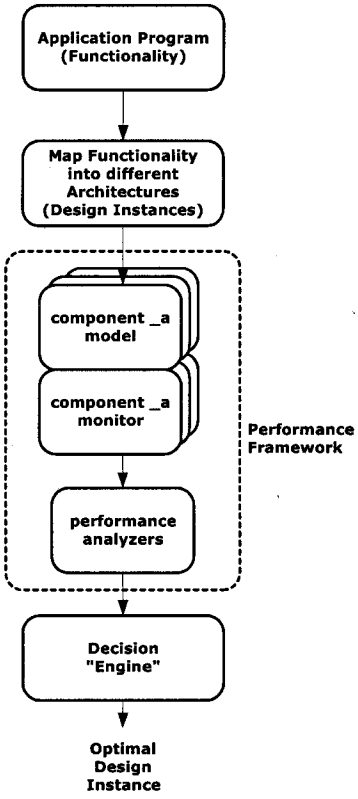

Figure 6-2 Performance Framework and Decision Engine integration

6.2 Performance Evaluation

In order to assess the quality of different design instances we follow an approach consisting of six steps: 1) defining a proper set of parameters (viz., performance metrics) upon which to base the design assessment, 2) modeling the application's functionality 3) modeling different architectures, 4) partitioning the functionality and mapping it into the different architectures, 5) simulating the application by executing the models, and 6) applying MCDM methods to rank the different design instances. Figure 6-3 summarizes the conceptual structure of the approach.

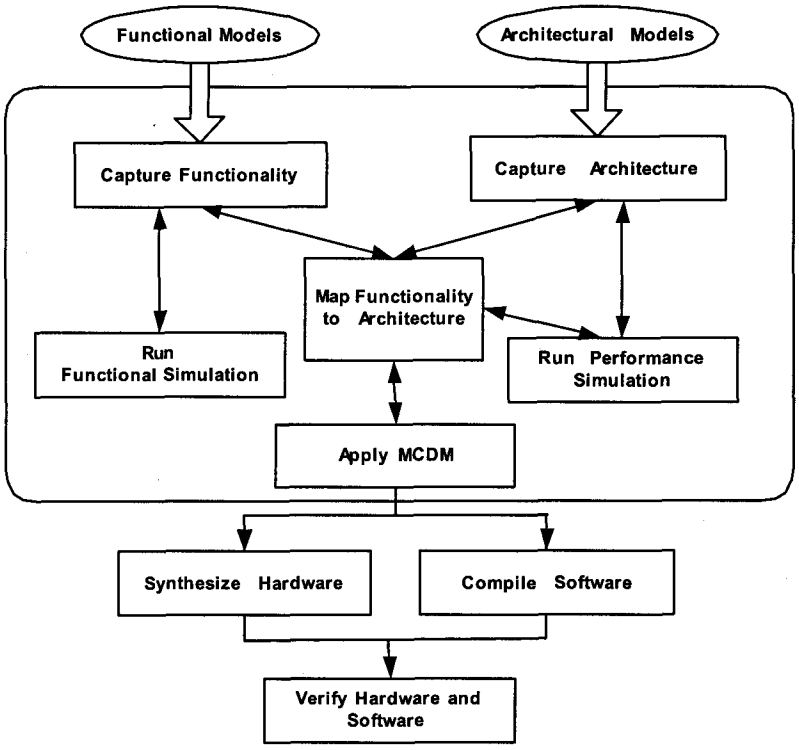

Figure 6-3 Performance Evaluation Approach

As shown in Figure 6-4 the functional description of an embedded system can be partitioned in many different ways, and mapped into several different architectures. Given a functionality-architecture mapping M_i and an input stimulus we can execute the associated model, and by properly defining n probes, measure the values of the n performance metrics of interest. We can differentiate the importance to the various performance metrics p_i by assigning them different weights w_i. Putting together the performance measures obtained for all possible mappings we can form a decision matrix and use MCDM methods to rank the different mappings [80]. Element a_{ij} of the decision matrix A represents the value of the j-th performance metric for the i-th mapping.

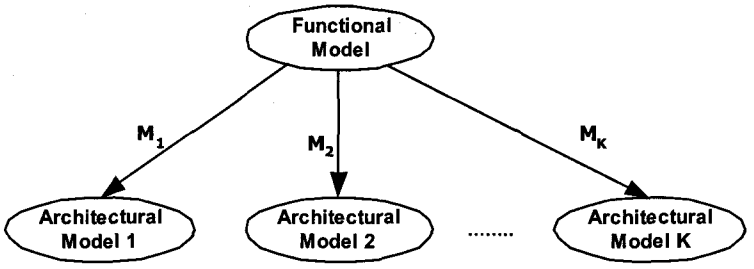

Figure 6-4 Function-Architecture mapping

Figure 6-5 provides a pictorial representation of these ideas.

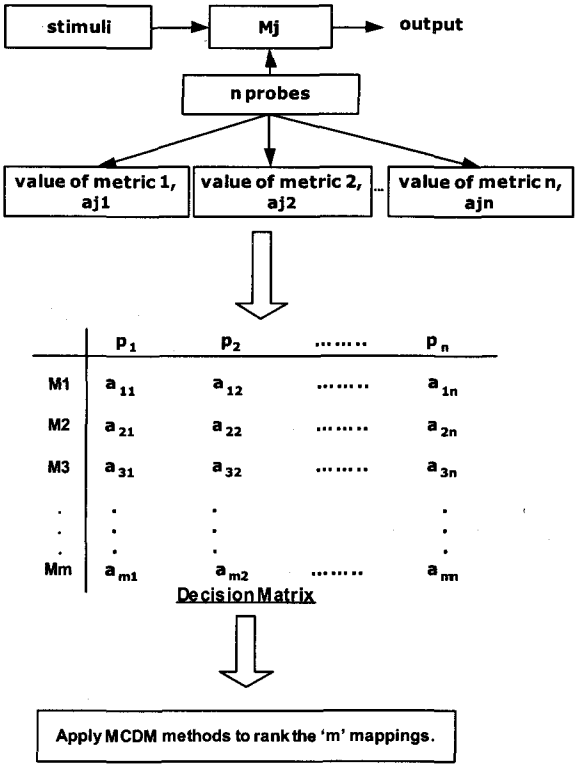

Figure 6-5 Design Methodology

6.3 Multi-Criteria Decision Making Methods

Multi-Criteria Decision Making (MCDM) is a class of methods for making optimal multi-objective decisions. Given a set of alternatives and a set of decision criteria, MCDM methods can be used to find the best alternative. In cases in which the number of criteria is large, criteria may be arranged in a hierarchical manner. Sometimes, in literature the "decision criteria" are also referred as "attributes" or "goals". Since different criteria represent different dimensions of the alternatives, they may conflict with each other. Different criteria may also be associated with different units of measures, making MCDM problems intrinsically hard to solve [81]. MCDM methods can be

applied on deterministic, fuzzy, and stochastic data. In our case the data used are deterministic and the set of alternatives is finite. There is no single MCDM method that outperforms the others in all aspects. Thus, invariably a comparative approach is used when dealing with MCDM methods.

In the case of embedded systems the focus is on discrete decision problems, with conflicting criteria. MCDM methods are used to rank the alternatives at hand based on their relative quality with respect to the chosen criteria. The alternatives are the possible mappings of the functional blocks of a system into hardware (ASICs) and software. The criteria used to rank the mappings are a set of performance metrics that capture both hardware and software characteristics.

Given a set of m alternatives represented as M_1, M_2,....M_m to be ranked using n decision criteria p_1, p_2,....p_n, we can form a decision matrix A of order (mxn) in which the element a_{ij} represents the value of the j-th criterion for the i-th alternative. MCDM assume that the values a_{ij} (i = 1,2....m and j = 1,2....n) are available in advance. Weights w_i for each decision criteria are assigned to represent relative importance.

6.3.1 The Weighted Sum Model (WSM)

The weighted sum model is the most commonly used method among the different MCDM methods. The WSM score of each alternative is the sum of the decision matrix value a_{ij} multiplied by the corresponding weight.

$$A_{iWSM-score} = \sum_{j=1}^{n} a_{ij} w_j, \quad \text{for } i = 1,2,....m \tag{6-1}$$

Different alternatives are compared on the basis of their scores and the best alternative is the one that has the highest (or lowest for minimization) WSM score.

6.3.2 The Weighted Product Model (WPM)

The weighted product model is very similar to the WSM. Here, the alternatives are compared by multiplying a number of ratios, one for each criterion. The ratio is raised to the power of the relative weight of the corresponding criterion. For example, to compare two alternatives M_K and M_L, the ratio R (M_K / M_L) is calculated as given by the Eq. (2).

$$R(M_K/M_L) = \prod_{j=1}^{n} (a_{Kj}/a_{Lj})^{wj} \quad (6\text{-}2)$$

If the ratio is greater than one, then the alternative M_K is better than M_L or vice versa. The best alternative is the one that is better than all other alternatives.

6.3.3 The Analytic Hierarchy Process (AHP)

The analytic hierarchy process (AHP) is a MCDM method which is performed following seven steps. This method is based on performing pair wise comparisons to determine the relative importance of each alternative in terms of each criterion. The algorithm for performing this method is as follows:

1) For each of the n criteria, a pair wise comparison matrix of order m×m is built (m is the number of alternatives). Each element a_{ij} of the matrix represents the pair wise comparison of alternative i-th and j-th. The value a_{ij} quantifies the relative importance of alternative Mi compared with Mj. The value is chosen based on a scale from 1 to 9, where 1 means equally preferred and 9 means extremely preferred [82]. We assign

these values based on relative percentages. The best alternative compared to the worst is given a value of 9 and all other comparisons are graded accordingly.

2) We compute the sum of each column of each comparison matrix.
3) Each element in the pair wise comparison matrices is divided by its column total.
4) We average the rows of the pair wise comparison matrices. As a consequence, for each matrix we get a priority vector of order mx1. There are n priority vectors, one for each pair wise comparison matrix.
5) A pair wise comparison matrix (nxn) is built out of the n performance criteria and its priority vector computed as described in steps 1-4.
6) The n priority vectors obtained in step 4 are formed into an mxn matrix and the matrix is multiplied with the priority vector found in step 5.
7) The resultant mx1 matrix represents the AHP scores of the m alternatives. The sum of these scores should be 1. Based on either maximization or minimization, the ranking is done.

6.4 A Design Example

To illustrate the application of MCDM methods on the design of embedded systems we apply it to the design of an Aircraft Pressurization System [64]. As altitude increases it becomes increasingly difficult for humans to handle the air pressure. Atmospheric pressure decreases as altitude increases as illustrated in Figure 6-6, therefore, in aircrafts flying at high altitudes, the cabin pressure has to be controlled. The APS is an automated control system that controls the pressure in an aircraft cabin within comfortable limits. If

pressure control is not provided, many physiological problems such as ear-ache and gastro-intestinal problems, may occur.

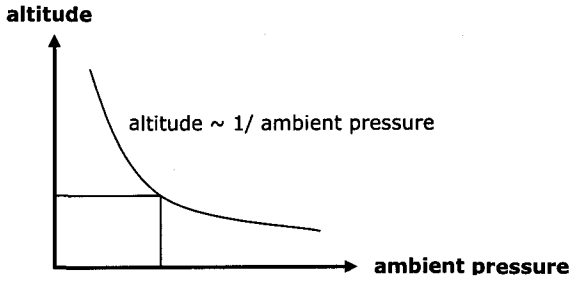

Figure 6-6 Altitude vs. ambient pressure

The inputs to the APS are ambient pressure, cabin pressure and a signal that indicates emergency. The outputs of the APS are signals that control three different valves namely pressure valve, release valve and dump valve. A high signal to any of these valves causes them to open while a low signal closes them. The pressure valve is used to pump air into the cabin to raise the cabin pressure. The release valve causes slow and steady release of cabin pressure. The dump valve is opened in case of emergency for a quick decay in cabin pressure. The inputs and outputs of the aircraft pressurization system are shown in Figure 6-7.

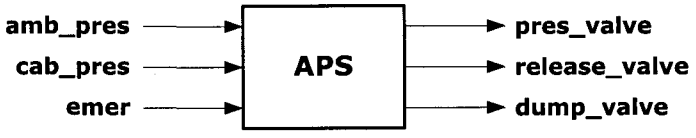

Figure 6-7 Basic block diagram of APS

The operation of the APS is illustrated by the detailed block diagram shown in Figure 6-8, and the finite state machine diagram in Figure 6-9. If the aircraft is flying at altitudes of less than 5000 ft., the cabin can be comfortably maintained at ambient pressure and the APS is said to be in *normal* state of operation. In *normal* state the pressure valve is closed and the release valve is kept open to equalize the cabin pressure to ambient pressure. As the aircraft flies to altitudes higher than 5000 ft, there is a fall in ambient pressure that makes it necessary to pressurize the cabin. The APS moves itself into *isobar* state to maintain a constant cabin pressure equivalent to the ambient pressure at 5000 ft, even when the real altitude is greater. In *isobar* state, if necessary, the pressure valve is opened and the release valve remains closed. When the altitude of the aircraft reaches 24000 ft, the APS changes its state of operation from *isobar* to *differential*. In *differential* state the APS maintains a differential of 6.5 psi between cabin pressure and ambient pressure. This is accomplished by either opening or closing the pressure or release valves.

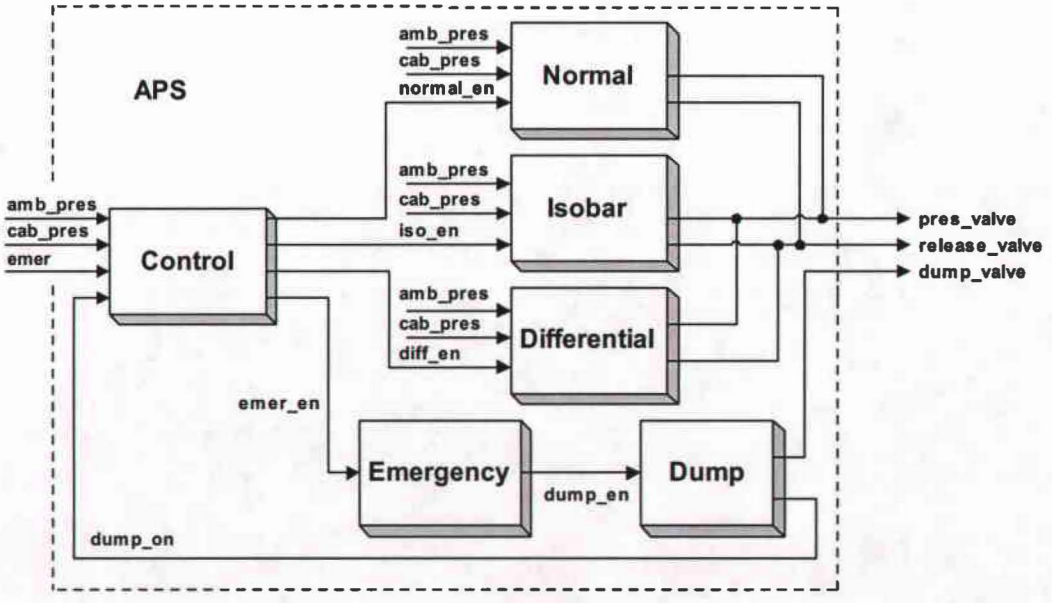

Figure 6-8 Detailed block diagram of APS

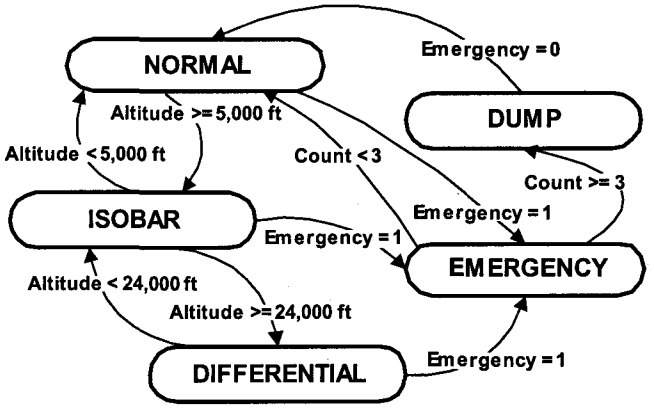

Figure 6-9 Finite state machine diagram of APS

The APS should be able to handle an emergency situation of excessive cabin pressure during any state of operation. The APS goes into *emergency* mode if the emergency signal is high. The emergency block checks if the emergency was real or just a glitch. If the emergency signal remains high continuously for 3 clock cycles the APS goes to *dump* state, else APS returns to *normal* state to resume operation. In the *dump* state the dump valve is opened to cause a quick decay of the cabin pressure. Once the cabin pressure becomes less than or equal to ambient pressure and the emergency signal goes low, the APS goes back to *normal* state of operation.

6.5 Experimental Setup and Results

For each mapping we assume that the target architecture consists of two ASICs based on different technologies and a software module running on a microcontroller. The six blocks forming the APS can be implemented in $3^6 = 729$ combinations. In order to demonstrate our methodology we considered only 8 different mappings (Table 6-1). An attempt has been made to choose the 8 mappings as representative as possible. Four

performance metrics have been used to assess the quality of the 8 different mappings of the APS. The four performance metrics considered are: *execution time, utilization, area,* and *program memory*. Execution time is the average time required by the system to execute a test scenario from start to finish. Utilization is the ratio of the used resources (both hardware and software) to the available resources. Area represents the amount of silicon required if a functional block is implemented in hardware, while program memory represents the amount of memory that would be required to store the program if a functional block is implemented in software. Four performance metrics have been used to assess the quality of the 8 different mappings of the APS. The four performance metrics considered are: *execution time, utilization, area,* and *program memory*. Execution time is the average time required by the system to execute a test scenario from start to finish. Utilization is the ratio of the used resources (both hardware and software) to the available resources. Area represents the amount of silicon required if a functional block is implemented in hardware, while program memory represents the amount of memory that would be required to store the program if a functional block is implemented in software.

Mappings	Control	Normal	Isobar	Differential	Dump	Ememgency
1	ASIC1	ASIC2	ASIC1	ASIC2	ASIC2	ASIC1
2	ASIC1	SW	ASIC1	SW	ASIC1	SW
3	SW	ASIC2	SW	ASIC2	SW	ASIC2
4	SW	ASIC1	ASIC2	ASIC2	SW	SW
5	ASIC2	SW	SW	SW	ASIC2	ASIC1
6	ASIC1	SW	ASIC2	SW	SW	ASIC1
7	SW	ASIC2	SW	ASIC1	ASIC2	SW
8	SW	SW	SW	SW	SW	SW

Table 6-1 Different Mappings

The test case scenario and the probes to monitor the response to evaluate the system performance have been modeled in SystemC. The complexity of the functional units used to implement the APS has been represented in terms of gate count in case of hardware

implementation or memory size in case of software implementation. Since program memory is the amount of memory required to store the executable of a functional block, memory size is directly proportional to the SystemC object file. Simulation was done and the values of the performance metrics of all the mappings were found. The values are summarized in Table 6-2. All metrics except *utilization* need minimization. Therefore, for consistency reasons we decided to consider the inverse of *utilization*. MCDM methods were applied on the data summarized in Table 6-2.

Mappings	Execution Time x 10 (s)	Program Memory x 10 (KB)	Area x 10^3 $(\mu m)^2$	1/Utilization x 10^{-3}
w_i	**0.4**	**0.15**	**0.10**	**0.35**
1	18	0	18.9	25
2	27	18	7.3	26
3	33	21	8.3	27.9
4	22.5	20	9.8	25.5
5	40.5	19	10.7	26.2
6	27.5	13	7.5	33.4
7	30	19	9.7	26.2
8	42	39	0	50

Table 6-2 Decision Matrix

WPM scores for the comparison of all possible combinations of the 8 mappings are shown in Table 6-3.

R(M1/M2)	0	R(M3/M4)	1.1917
R(M1/M3)	0	R(M3/M5)	0.9321
R(M1/M4)	0	R(M3/M6)	1.0963
R(M1/M5)	0	R(M3/M7)	1.0613
R(M1/M6)	0	R(M3/M8)	∞
R(M1/M7)	0	R(M4/M5)	0.7821
R(M1/M8)	0	R(M4/M6)	0.9200
R(M2/M3)	0.8685	R(M4/M7)	0.8906
R(M2/M4)	1.0350	R(M4/M8)	∞
R(M2/M5)	0.8096	R(M5/M6)	1.1762
R(M2/M6)	0.9522	R(M5/M7)	1.1386
R(M2/M7)	0.9218	R(M5/M8)	∞
R(M2/M8)	∞	R(M6/M7)	0.9680
		R(M6/M8)	∞
		R(M7/M8)	∞

Table 6-3 WPM results

Table 6-4 gives the scores of the mappings and their rankings based on the three MCDM methods applied. Most of the MCDM methods rank M1 as the best mapping. The mappings when arranged in decreasing order are:

M1>M4>M2>M7>M6>M3>M5>M8

As expected, the best performance is obtained when complete hardware implementation is considered. Mapping M1 does not use at all software modules. Its nearest alternative M4 instead uses both hardware and software modules.

Mapping	WSM score & rank		AHP score & rank		WPM rank
1	17.84	1	0.1253	6	1
2	23.33	3	0.1234	3	3
3	26.94	6	0.1248	5	6
4	21.9	2	0.0973	2	2
5	29.29	7	0.1913	8	7
6	25.39	5	0.0792	1	4
7	24.99	4	0.1271	4	5
8	40.15	8	0.1273	7	8

Table 6-4 Summary of rankings

6.6 Summary

In this chapter, we have introduced a possible methodology to perform design space exploration of an embedded system. The approach consists of: 1) modeling functionality and architecture of the system, 2) map the functionality into several different architectures 3) run functional and performance simulations and 4) rank the different design choices using MCDM. The approach has been validated by applying it to the design of an Aircraft Pressurization System.

CHAPTER 7
CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

In this work, we have introduced a system level approach for the early design assessment of embedded system performance. The distinguishing feature of the approach is to derive system performance by executing high level models rather than gate level or transistor level pre characterizations. This allows assessing performance much earlier in the design cycle and leads to faster execution times. Due to the growing popularity of portable computing devices, the performance index we have focused on is power consumption.

First, we have described previous work in power estimation and our system-level power estimation approach. We propose a technique in which the power figures associated with each component of the system are derived through a four steps procedure. The steps are: 1) translating each core's functionality to a set of primitive instructions, 2) simulating the application program, 3) mapping the instructions requested by the application program into abstract functional units, and 4) computing the aggregate power consumption of the entire system. Second, we have described the implementation details of the framework developed to realize the methodology proposed. The framework has been implemented using SystemC. The choice of SystemC permits the use of the same language (C++) for describing both hardware and software components. Besides providing a common high level language, SystemC can also be linked to commercial tools for hardware synthesis. Third, we have demonstrated the feasibility and accuracy of the methodology by applying it to different design examples. We have compared the

results against gate level simulation, and actual measurements on the real hardware. The results of our system-level approach are within 10% of the gate level and physical measurement. Lastly, we have presented a technique for finding among several design alternatives the one that optimally solves the simultaneous satisfaction of multiple objectives. Design space exploration is based on the use of Multi-Criteria Decision Methods (MCDM). The methodology consists of: 1) modeling functionality and architecture of the system, 2) mapping the functionality into several different architecture, 3) running functional and performance simulations, and 4) ranking the different design instances using MCDM.

7.2 Future Work

High level tools fill a significant gap in current design methodologies, however much remains to be done in order to demonstrate the robustness and applicability of the techniques in a realistic industrial setting. Future work in this area needs the development of algorithms that can reduce the computational complexity of problems such as design space exploration, hardware/software synthesis optimization, dynamic management of power aware systems, and automatic system partitioning. Future work can also concentrate on the development of compilers to automate the translation process from system specification to performance assessment, and to extend them to as many performance indices as possible (examples include power consumption, execution time, silicon area, and software size).

APPENDIX A

This appendix collects the energy profiles of the twenty application programs used to benchmark our power estimation framework, and the average current measured for each CPU instruction.

A.1 Energy Plots

Figure A–1 through Figure A–20 show the energy consumed by the system during the execution of the twenty application programs benchmarked. The capability of profiling energy is particularly useful to gain insight into the system hot spots.

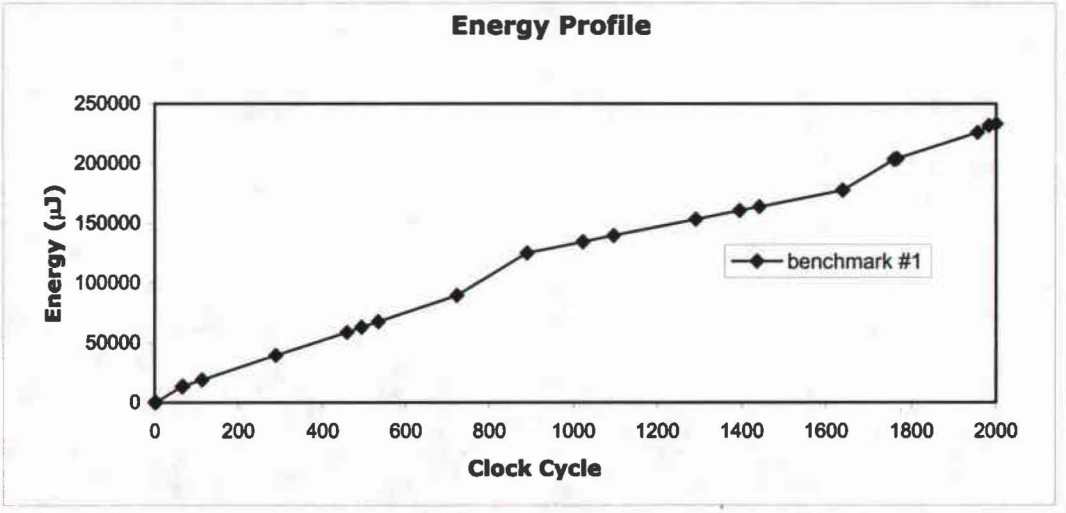

Figure A–1 Plot of energy per clock cycles elapsed for benchmark #1

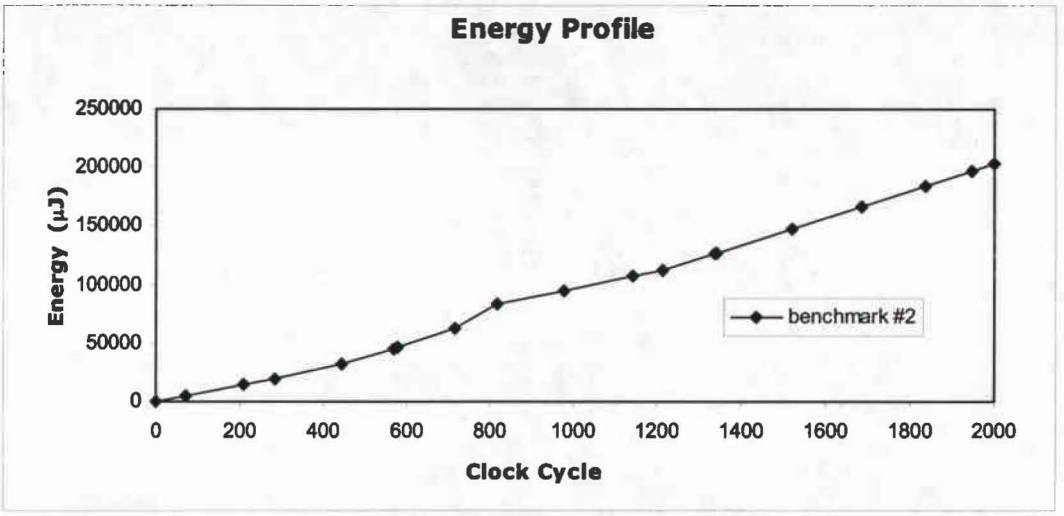

Figure A–2 Plot of energy per clock cycles elapsed for benchmark #2

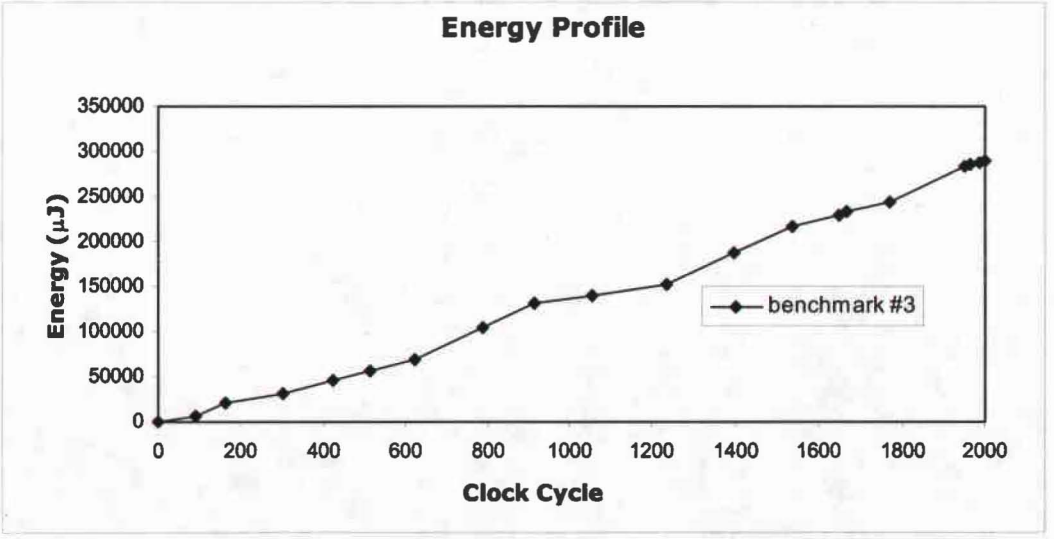

Figure A–3 Plot of energy per clock cycles elapsed for benchmark #3

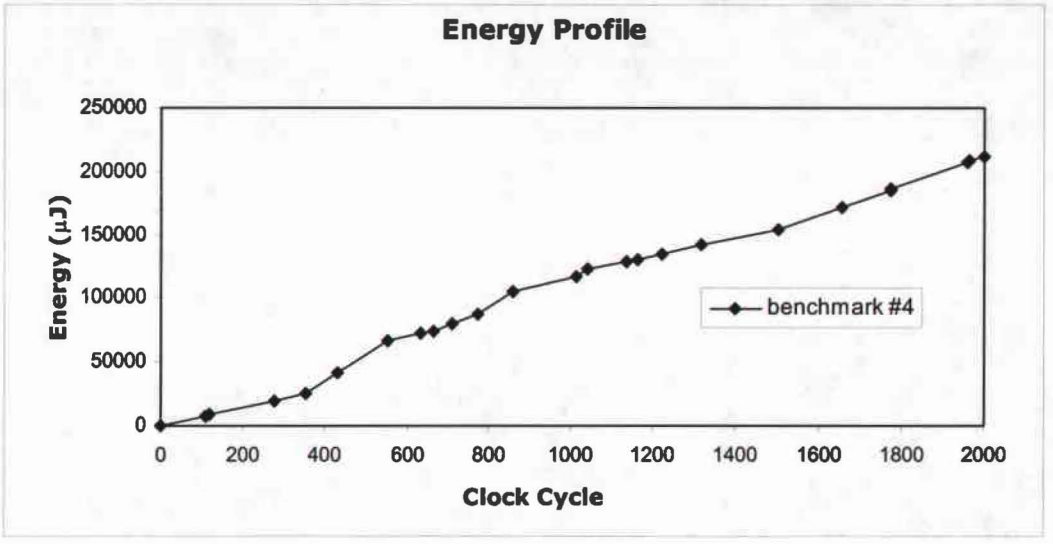

Figure A–4 Plot of energy per clock cycles elapsed for benchmark #4

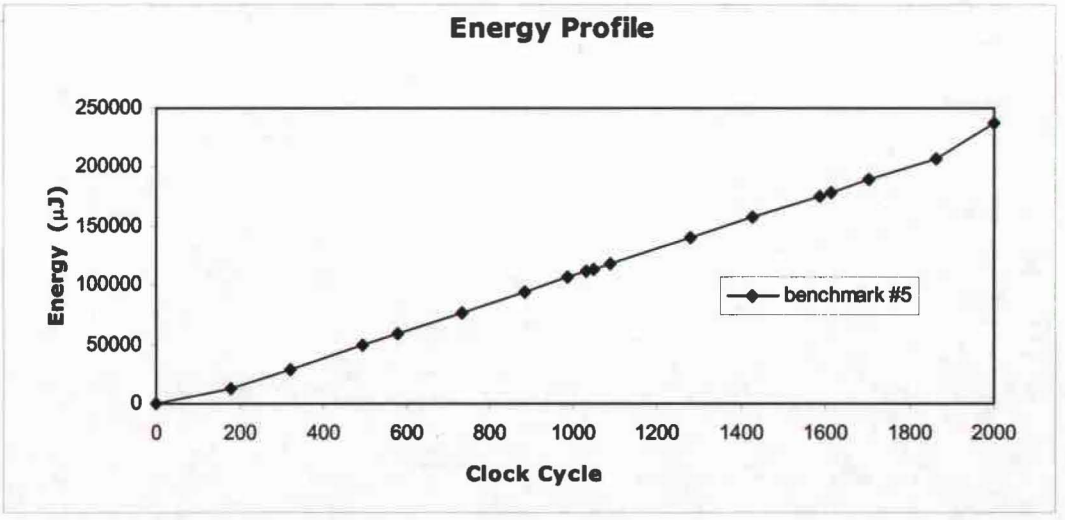

Figure A–5 Plot of energy per clock cycles elapsed for benchmark #5

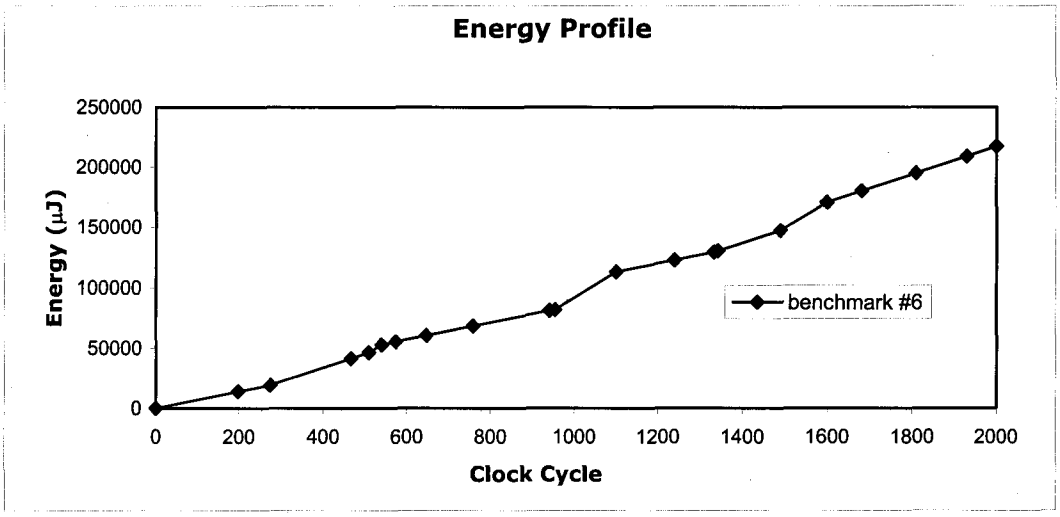

Figure A–6 Plot of energy per clock cycles elapsed for benchmark #6

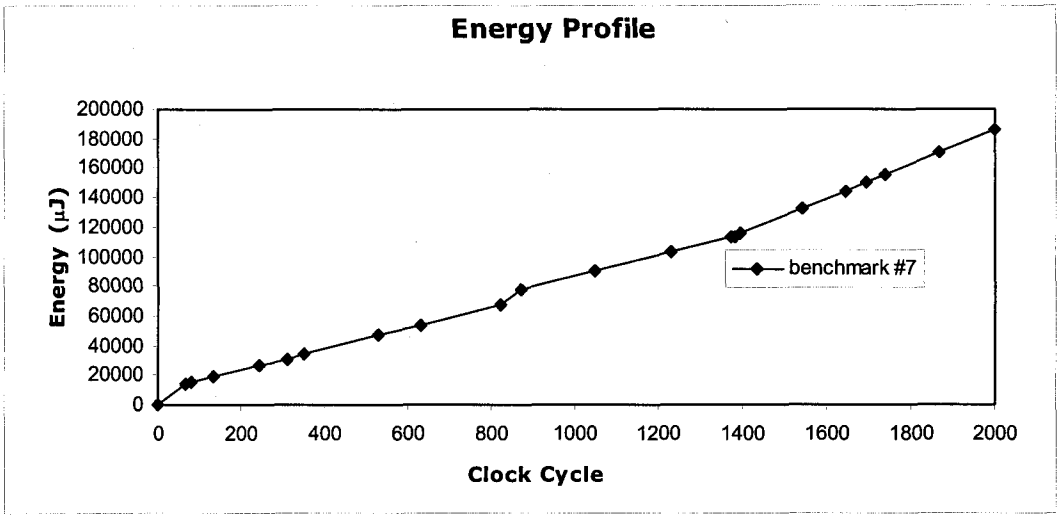

Figure A–7 Plot of energy per clock cycles elapsed for benchmark #7

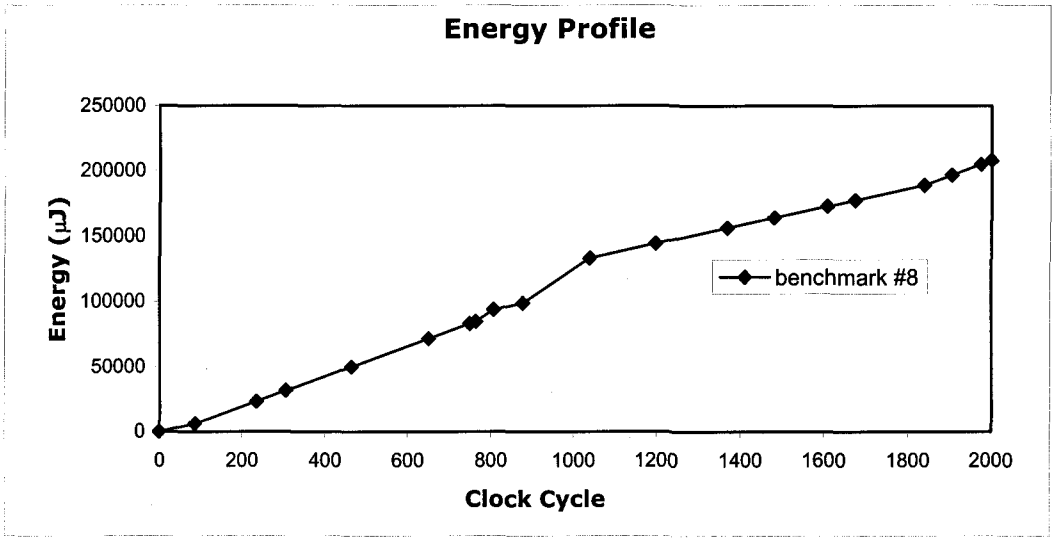

Figure A–8 Plot of energy per clock cycles elapsed for benchmark #8

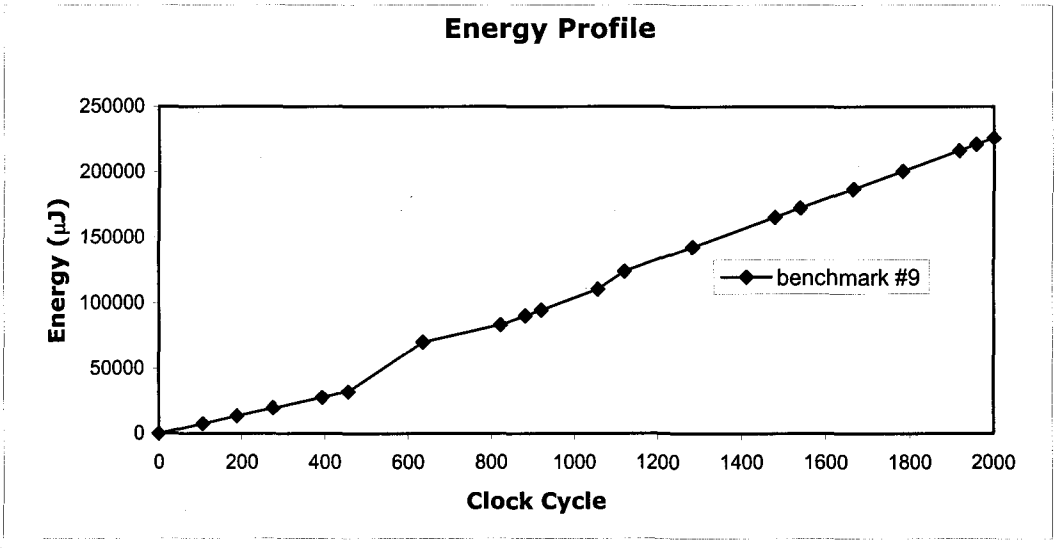

Figure A–9 Plot of energy per clock cycles elapsed for benchmark #9

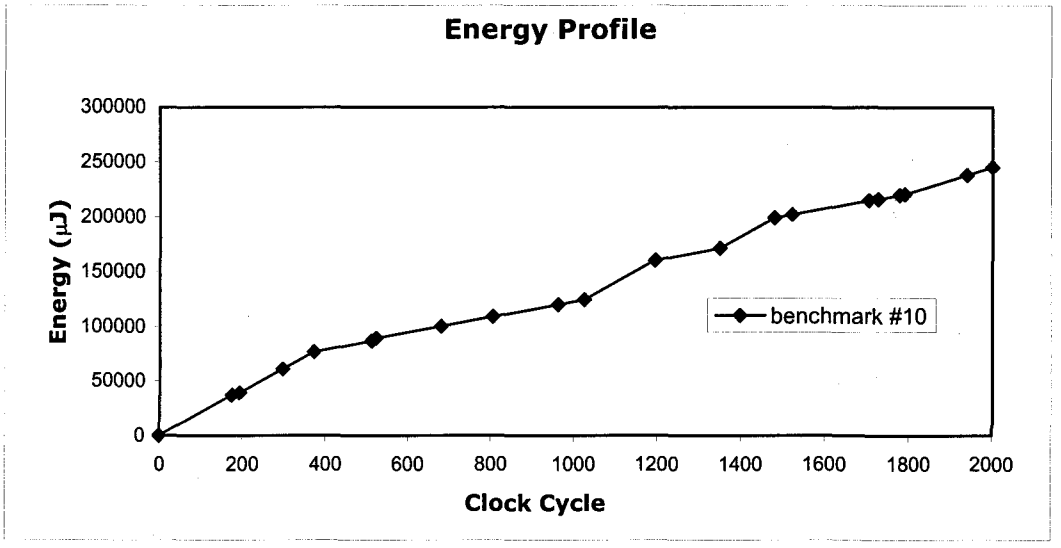

Figure A–10 Plot of energy per clock cycles elapsed for benchmark #10

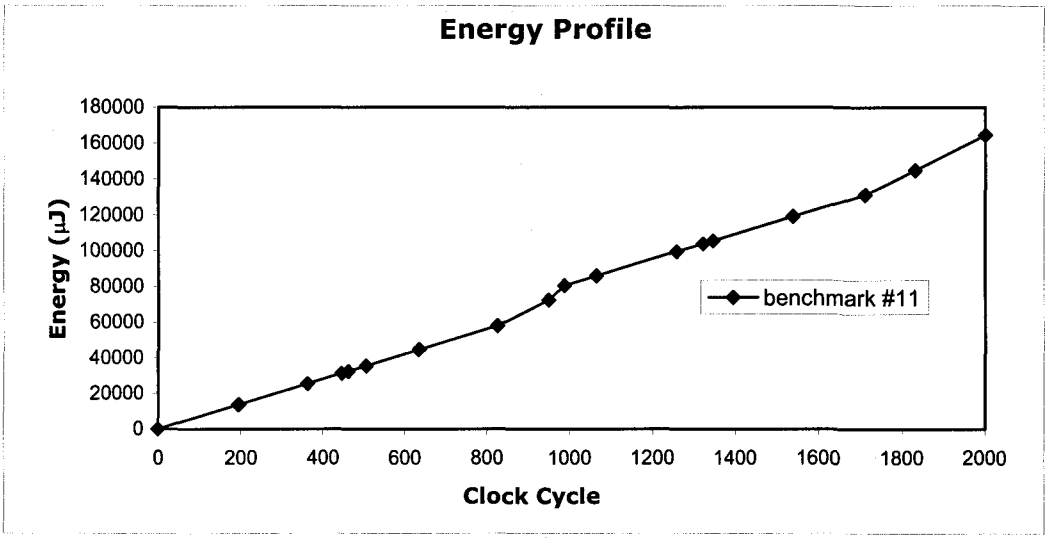

Figure A–11 Plot of energy per clock cycles elapsed for benchmark #11

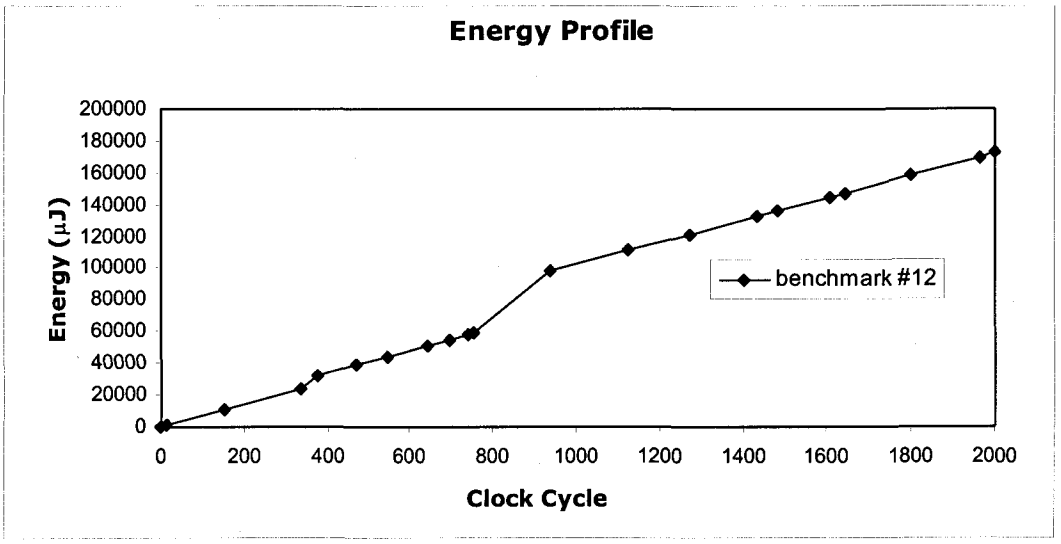

Figure A–12 Plot of energy per clock cycles elapsed for benchmark #12

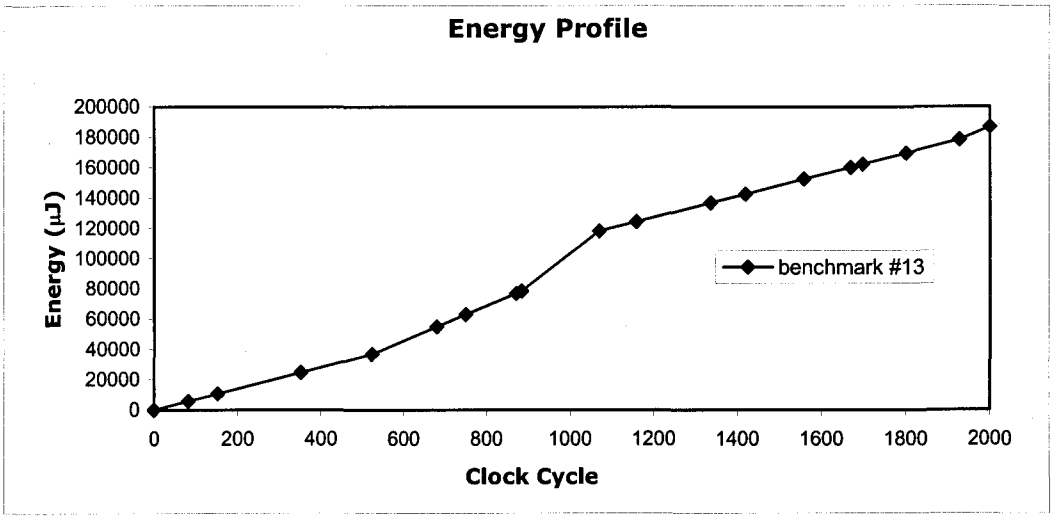

Figure A–13 Plot of energy per clock cycles elapsed for benchmark #13

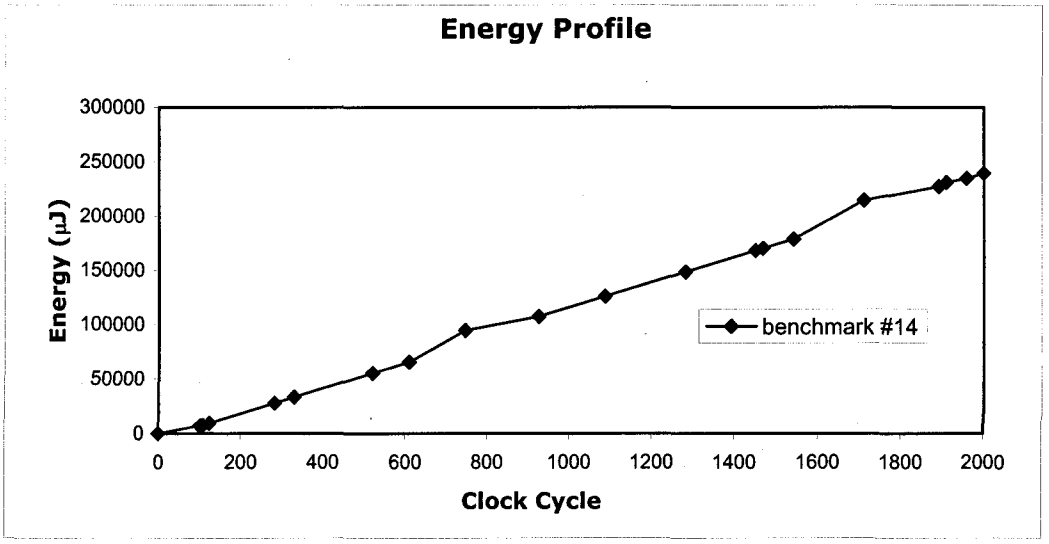

Figure A–14 Plot of energy per clock cycles elapsed for benchmark #14

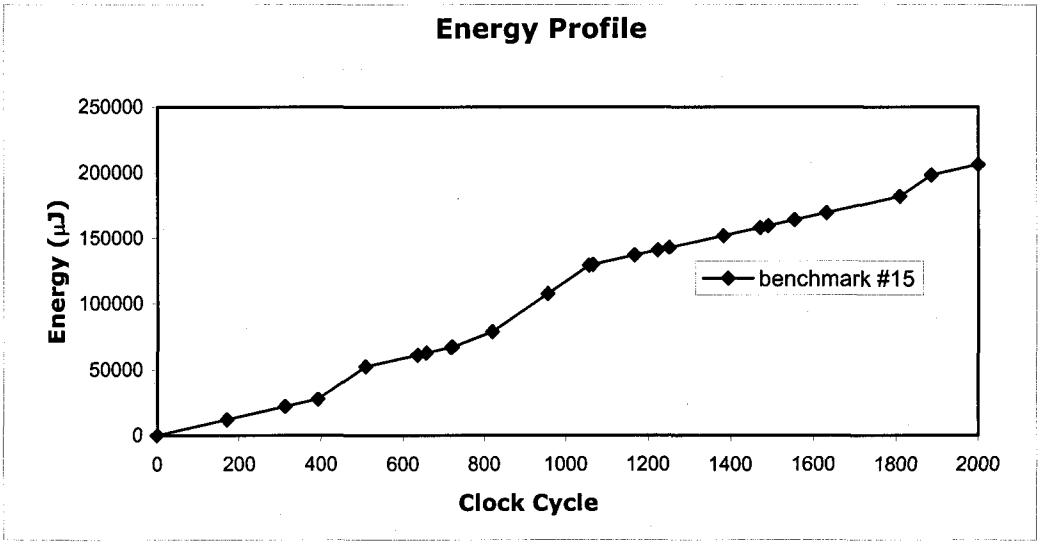

Figure A–15 Plot of energy per clock cycles elapsed for benchmark #15

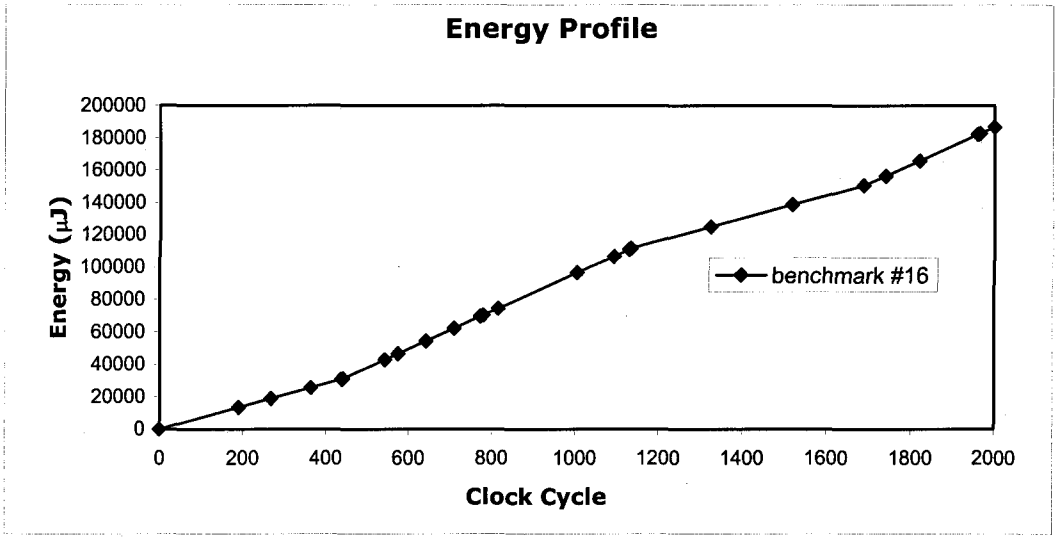

Figure A–16 Plot of energy per clock cycles elapsed for benchmark #16

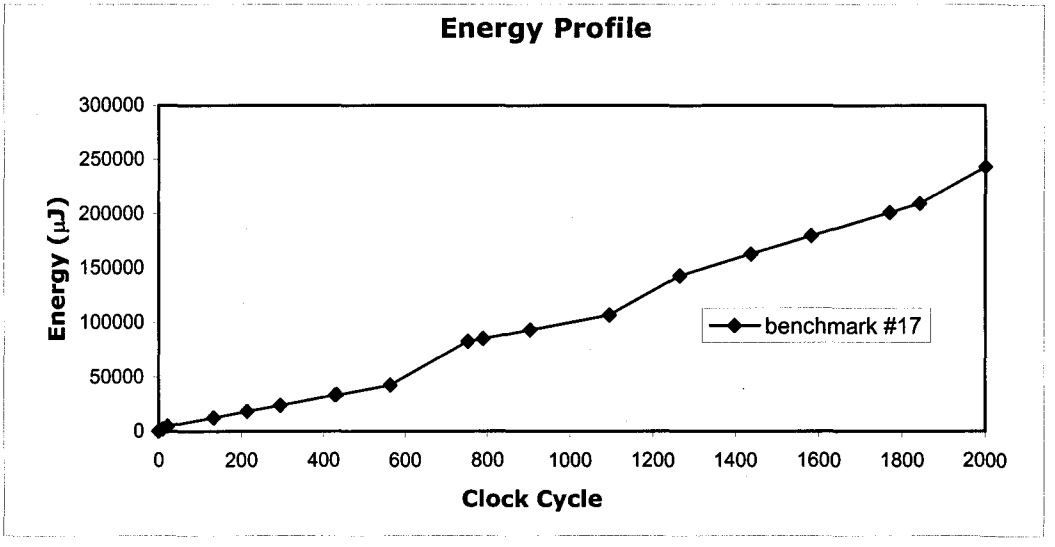

Figure A–17 Plot of energy per clock cycles elapsed for benchmark #17

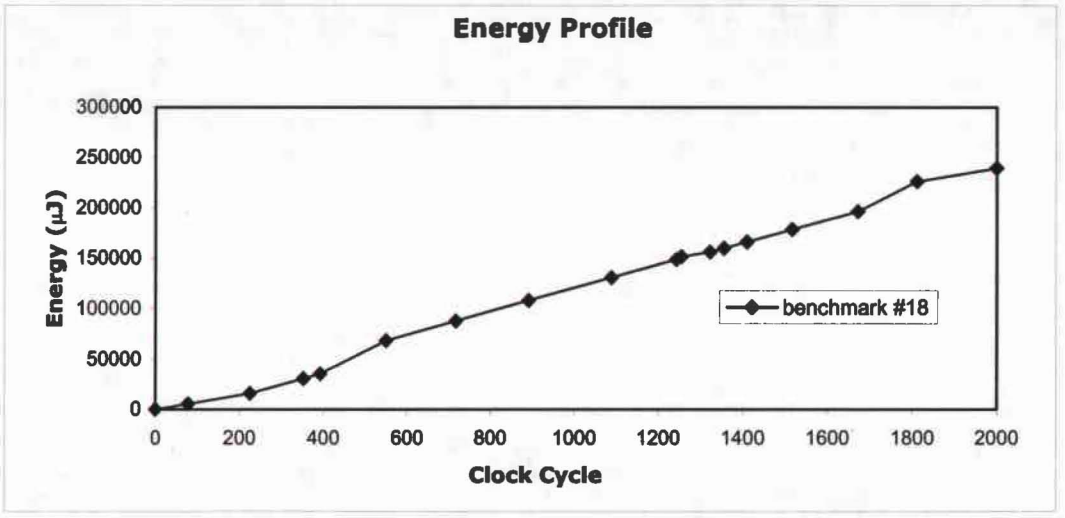

Figure A–18 Plot of energy per clock cycles elapsed for benchmark #18

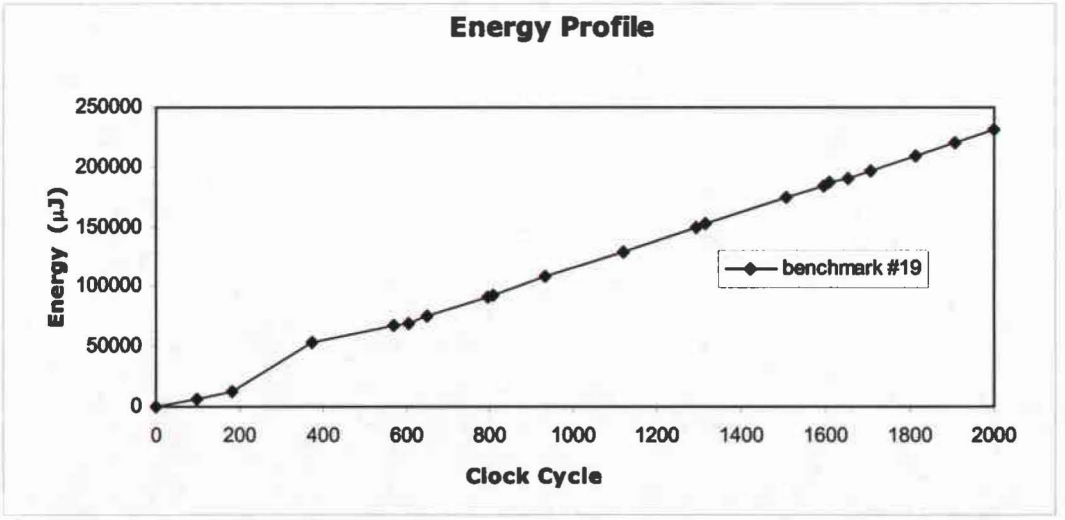

Figure A–19 Plot of energy per clock cycles elapsed for benchmark #19

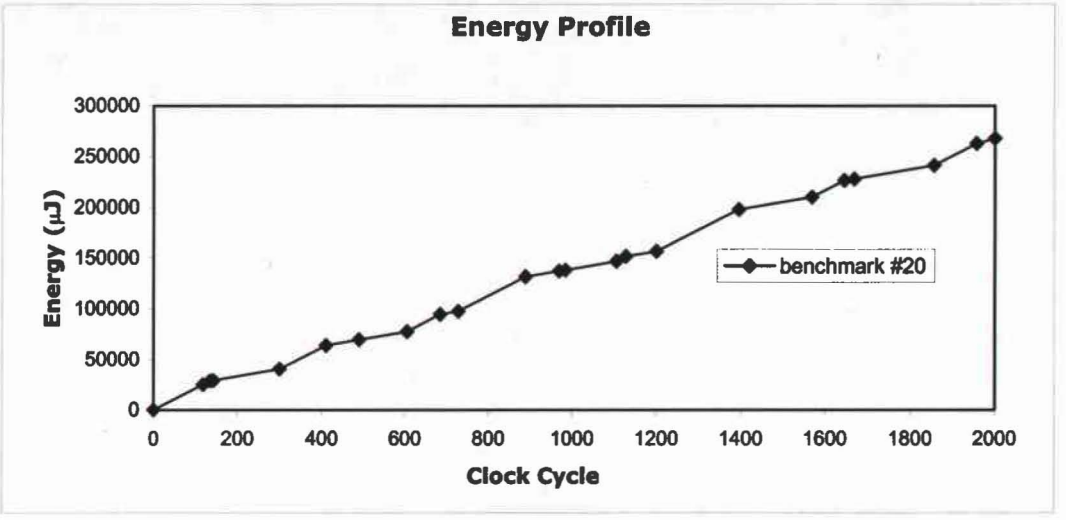

Figure A–20 Plot of energy per clock cycles elapsed for benchmark #20

A.2 Average current

Table A-1 summarizes the average current absorbed by the most commonly used instructions of the CPU.

Number	Instruction	Bytes	Average Current (mA)
1	MOV reg, #data16	4	11.34124
2	MOVB reg, #data8	4	10.91897
3	MOV mem, reg	4	11.30886
4	MOVB mem, reg	4	11.30199
5	BCLR bitoff.1	2	10.92310
6	BSET bitoff.1	2	10.93395
7	MOV Rw, Rw	2	10.95270
8	MOVB Rb, Rb	2	11.02427
9	MOV reg, mem	4	11.82794
10	MOVB reg, mem	4	11.71842
11	JMPS seg, caddr	4	11.67490
12	RETI	2	9.764218
13	ADD Rw, Rw	2	11.32220
14	BCLR bitoff.3	2	10.92248
15	SUB Rw, [Rw]	2	11.64922
16	MOV Rw, #data4	2	11.14936
17	MOV [Rw], Rw	2	11.63738
18	MOV [Rw+#data16], Rw	4	11.59374
19	BSET bitoff.4	2	10.92256
20	BMOVN bitaddr, bitaddr	4	11.16201
21	JMPR cc_UC, rel	2	9.488694
22	MOV Rw, [Rw]	2	11.79919
23	MOV Rw, [Rw+#data16]	4	11.78101
24	CMP Rw, [Rw]	2	11.19566
25	JMPR cc_UGT, rel	2	9.475835
26	ADDB Rb, Rb	2	10.98539
27	ADD reg, mem	4	11.92048
28	ADDB reg, mem	4	11.47500
29	ADD mem, reg	4	11.07846
30	ADDB mem, reg	4	11.01743
31	ADD reg, #data16	4	12.01463
32	ADDB reg, #data8	4	11.06560
33	ADD Rw, [Rw]	2	11.62645
34	ADDB Rb, [Rw]	2	11.32223
35	BCLR bitoff.0	2	10.88539
36	BSET bitoff.0	2	10.90749
37	ADDC Rw, Rw	2	11.83624
38	ADDCB Rb, Rb	2	10.97740

39	ADDC reg, mem	4	11.91078
40	ADDCB reg, mem	4	11.46813
41	ADDC mem, reg	4	11.08427
42	ADDCB mem, reg	4	11.05469
43	ADDC reg, #data16	4	11.96423
44	ADDCB reg, #data8	4	11.02958
45	ADDC Rw, [Rw]	2	11.60826
46	ADDCB Rb, [Rw]	2	11.30533
47	JMPR cc_NET, rel	2	9.482636
48	SUB Rw, Rw	2	11.90550
49	SUBB Rb, Rb	2	11.03380
50	SUB reg, mem	4	11.92659
51	SUBB reg, mem	4	11.51307
52	SUB mem, reg	4	11.08947
53	SUBB mem, reg	4	11.03845
54	SUB reg, #data16	4	12.02207
55	SUBB reg, #data8	4	11.09549
56	SUBB Rb, [Rw]	2	11.36029
57	BCMP bitaddr, bitaddr	4	11.07649
58	JMPR cc_Z, rel	2	9.490272
59	BCLR bitoff.2	2	10.92624
60	BSET bitoff.2	2	10.92928
61	SUBC Rw, Rw	2	11.94794
62	SUBCB Rb, Rb	2	11.03091
63	SUBC reg, mem	4	11.94756
64	SUBCB reg, mem	4	11.48454
65	SUBC mem, reg	4	11.10488
66	SUBCB mem, reg	4	11.04340
67	SUBC reg, #data16	4	11.96791
68	SUBCB reg, #data8	4	11.12920
69	SUBC Rw, [Rw]	2	11.67714
70	SUBCB Rb, [Rw]	2	11.35856
71	JMPR cc_NZ, rel	2	9.497392
72	BSET bitoff.3	2	10.95938
73	CMP Rw, Rw	2	10.91408
74	CMPB Rb, Rb	2	11.05441
75	CMP reg, mem	4	11.20839
76	CMPB reg, mem	4	11.29377
77	CMP reg, #data16	4	10.96454
78	CMPB reg, #data8	4	10.95464
79	CMPB Rb, [Rw]	2	11.28362
80	BMOV bitaddr, bitaddr	4	10.74283
81	JMPR cc_V, rel	2	9.467650
82	BCLR bitoff.4	2	10.93704
83	XOR Rw, Rw	2	11.65051
84	XORB Rb, Rb	2	10.97710
85	XOR reb, mem	4	11.68422

86	XORB reg, mem	4	11.36560
87	XOR mem, reg	4	11.04226
88	XORB mem, reg	4	11.03719
89	XOR reg, #data16	4	11.75769
90	XORB reg, #data8	4	11.04372
91	XOR Rw, [Rw]	2	11.47816
92	XORB Rb, [Rw]	2	11.28153
93	BOR bitaddr, bitaddr	4	10.08217
94	JMPR cc_NV, rel	2	9.487355
95	BCLR bitoff.5	2	10.93873
96	BSET bitoff.5	2	10.93654
97	AND Rw, Rw	2	10.95496
98	ANDB Rb, Rb	2	11.03939
99	AND reg, mem	4	11.51890
100	ANDB reg, mem	4	11.32372
101	AND mem, reg	4	10.01253
102	ANDB mem, reg	4	11.06110
103	AND reg, #data16	4	11.29019
104	ANDB reg, #data8	4	11.10327
105	AND Rw, [Rw]	2	11.54926
106	ANDB Rb, [Rw]	2	11.45762
107	BAND bitaddr, bitaddr	4	10.79008
108	JMPR cc_N, rel	2	9.505215
109	BCLR bitoff.6	2	10.96748
110	BSET bitoff.5	2	10.96440
111	OR Rw, Rw	2	11.08677
112	ORB Rb, Rb	2	11.08423
113	OR reg, mem	4	11.48257
114	ORB reg, mem	4	11.42748
115	OR mem, reg	4	11.08256
116	ORB mem, reg	4	11.06998
117	OR reg, #data16	4	10.02741
118	ORB reg, #data8	4	10.97589
119	OR Rw, [Rw]	2	11.49967
120	ORB Rb, [Rw]	2	11.41494
121	BXOR bitaddr, bitaddr	4	11.22040
122	JMPR cc_NN, rel	2	9.505441
123	BCLR bitoff.7	2	10.96973
124	BSET bitoff.8	2	10.98463
125	MOV [Rw], mem	4	11.45932
126	IDLE	4	6.636212
127	MOV [-Rw], Rw	2	10.84807
128	MOVB [-Rw], Rb	2	10.74117
129	JMPR cc_C	2	9.491015
130	BCLR bitoff.8	2	10.94398
131	BSET bitoff.8	2	10.96332
132	MOV mem, [Rw]	4	11.81612

133	PWRDN	4	0.007771
134	JMPR cc_NC	2	9.492706
135	BCLR bitoff.9	2	10.97997
136	BSET bitoff.9	2	10.96523
137	MOVB [Rw], mem	4	11.63141
138	MOVB Rb, [Rw]	2	11.56605
139	JMPR cc_SGT, rel	2	9.484365
140	BCLR bitoff.10	2	10.94732
141	BSET bitoff.10	2	10.94892
142	MOVB mem, [Rw]	4	11.65246
143	MOVB [Rw], Rb	2	11.47563
144	JMPR cc_SLE	2	9.498967
145	BCLR bitoff.11	2	10.96699
146	BSET bitoff.11	2	10.98513
147	MOV [Rw], [Rw]	2	11.87208
148	MOVB [Rw], [Rw]	2	11.78080
149	NOP	2	10.78512
150	JMPR cc_SLT	2	9.510617
151	BCLR bitoff.12	2	10.95858
152	BSET bitoff.12	2	10.94049
153	MOV [Rw+], [Rw]	2	16.02451
154	JMPR cc_SGE, rel	2	9.497127
155	BCLR bitoff.13	2	10.96155
156	BSET bitoff.13	2	10.97641
157	MOVB Rb, #data4	2	11.13986
158	MOVB [Rw + #data16], Rb	4	11.40450
159	MOV [Rw], [Rw+]	2	11.04437
160	MOVB [Rw], [Rw+]	2	11.05546
161	BCLR bitoff.14	2	10.96042
162	BSET bitoff.14	2	10.96891
163	MOVB Rb, [Rw + #data16]	4	11.52204
164	JMPR cc_ULE, rel	2	9.496466
165	BCLR bitoff.15	2	10.98076
166	BSET bitoff.15	2	10.96999

Table A-1 Average current for a representative subset of the CPU instruction set.

BIBLIOGRAPHY

[1] S. Edwards, L. Lavagno, E.A. Lee, and A. Sangiovanni-Vincentelli. "Design of Embedded Systems: Formal Models, Validation, and Synthesis," *Proceedings of the IEEE*, vol. 85, no. 3, March 1997, pp. 366-390.

[2] M. Nemani and F.N. Najm, "High Level Area and Power Estimation for VLSI Circuits," *IEEE Trans. CAD of Integrated Circuits and Systems*, vol. 18, no. 6, June 1999, pp. 697-713.

[3] K. Keutzer, S. Malik, R. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli, "System-Level Design: Orthogonalization of Concerns and Platform-Based Design," *IEEE Trans. CAD of Integrated Circuits and Systems*, vol. 19, no. 12, Dec. 2000, pp. 1523-1543.

[4] S.Schulz and J.W.Rozenblit, "Refinement of Model Specifications in Embedded System Design," *Proc. Int'l Conf. and Workshop Engineering of Computer-Based Systems* (ECBS 02), IEEE Press, Apr. 2002, pp.159-166.

[5] P. Landman, "High-Level Power Estimation," *Proc. Int'l Symp. Low Power Electronics and Design* (ISLPED 96), IEEE Press, Aug. 1996, pp.29-35.

[6] D. Rakhamatov, S. Vrudhula, and D.A. Wallach, "A Model for Battery Lifetime Analysis for Organizing Applications on a Pocket Computer", *IEEE Trans. VLSI Systems*, vol. 11, no. 6, Dec. 2003, pp. 1019-1030.

[7] A. Raghunathan, N.K. Jha, and S. Dey, *High-Level Power Analysis and Optimization*, Kluwer Academic, 1998.

[8] V. Tiwari, *Logic and System Design for Low Power Consumption*, doctoral dissertation, Dept. Electrical Eng., Princeton University, Princeton, 1996.

[9] http://www.aceshardware.com/

[10] http://www.energystar.gov/

[11] http://www.energy.gov/

[12] S. Hamilton, "Taking Moore's Law in the Next Century", *IEEE Computer*, Jan. 1999, pp.43-48.

[13] http://www.darpa.mil/MTO/fcrp/presentations/Overview.pdf

[14] http://www.semiconductors.philips.com/products/asic/soc/methodology/

[15] N.H.E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design: A system Perspective*, 2nd ed., Addison-Wesley, 1994.

[16] H.J.M. Veendrick, "Short-Circuit Dissipation of Static CMOS Circuitry and its Impact on the Design of Buffer Circuits," *IEEE J. Solid-State Circuits*, vol. 19, no. 4, Aug. 1984, pp. 468-473.

[17] A.P. Chandrakasan and R.W. Brodersen, *Low Power Digital CMOS Design*, Kluwer Academic, 1995.

[18] A. Bellaouar and M.I. Elmasry, *Low Power Digital VLSI Design - Circuits and Systems*, Kluwer Academic, 1995.

[19] A.P. Chandrakasan, S. Sheng, and R.W. Brodersen, "Low Power CMOS Digital Design," *IEEE J. Solid-State Circuits*, vol. 27, no. 4, Apr. 1992, pp. 473-484.

[20] J.M. Rabaey and M. Pedram, eds., *Low Power Design Methodologies*, Kluwer Academic, 1995.

[21] V. Tiwari, D. Singh, S. Rajgopal, G. Metha, R. Patel, and F. Baetz, "Reducing Power in High-performance Microprocessors," *Proc. 35th Ann. Design Automation Conference* (DAC 98), IEEE Press, 1998, pp. 732-737.

[22] M.R. Stan and W.P. Burleson, "Bus-Invert Coding for Low-Power I/O," IEEE Trans. VLSI Systems, vol. 3, no. 1, March 1995, pp. 49-58.

[23] H. Mizas, D. Soudris, S. Theoharis, G. Theodoridis, A. Thanailakis, and C.E. Goutis, *LPGD: A low Power Design Methodology/Flow and its Application to the Implementation of A DCS1800-GSM/DECT Modulator/demodulator*, European Low Power Initiative for Electronic System Design, tech. report ESPRIT 25256, Univ. Thrace, Greece, 1998.

[24] S.M. Kang, "Accurate Simulation of Power Dissipation in VLSI Circuits," *IEEE J. Solid-State Circuits*, Oct. 1986, vol. 21, no. 5, pp. 889-991.

[25] G.Y. Yacoub and W.H. Ku, "An Accurate Simulation Technique for Short-Circuit Power Dissipation Based on Current Component Isolation," *Proc. Int'l. Symp. Circuits and Systems*, IEEE Press, 1989, pp. 1157-1161.

[26] R. Tjarnstorm, "*Power* Dissipation Estimate by Switch level Simulation," *Proc. Int'l. Symp. Circuits and Systems*, IEEE Press, 1989, pp. 881-884.

[27] T. Givargis, *System-level Exploration for Pareto-Optimal Configurations in Parameterized System-on-a-Chip Architectures*, doctoral dissertation, Dept. Computer Sciences, Univ. California, Riverside, 2001.

[28] T.H. Krodel, "PowerPlay - Fast Dynamic Power Evaluation Based on Logic Simulation," *Proc. Int'l. Conf. Computer Aided Design*, IEEE Press, 1991, pp. 96-100.

[29] T. Givargis, F. Vahid, and J. Henkel, "Instruction-Based System-Level Evaluation of System-on-a-Chip Peripherals Cores," *IEEE Trans. VLSI Systems*, Dec. 2002, vol. 10, no. 6, pp. 856-863.

[30] K.D.Müller-Glaser, K.Kirsch, and K.Neusinger, "Estimating Essential Design Characteristics to support Project Planning for ASIC Design Management," *Proc. Int'l Conf. Computer Aided Design*, IEEE Press, Nov. 1991, pp.148-151.

[31] D.Liu and C.Svensson, "Power Consumption in CMOS VLSI Chips," *IEEE J. Solid State Circuits*, vol. 29, no. 6, June 1994, pp.663-670.

[32] K.M.Büyükşahin, F.N.Najm, "High-Level Power Estimation with Interconnect Effects," *Proc. Int'l Symp. Low Power Electronics and Design* (ISLPED 00), IEEE Press, 2000, pp.197-202.

[33] K-T Cheng and V. Agrawal, "An Entropy Measure for the Complexity of Multi-output Boolean Functions," *27th Design Automation Conference* (DAC 90), IEEE Press, 1990, pp. 302-305.

[34] D. Marculescu, R. Marculescu, and M. Pedram, "Information Theoretic Measures for Power Analysis," *IEEE Trans. CAD of Integrated Circuits and Systems*, vol. 15, no. 6, June 1996, pp. 599-610.

[35] A. Bogliolo and L. Benini, "Robust RTL Power Macromodels," *IEEE Trans. VLSI Systems*, vol. 6, no. 4, Dec. 1998, pp. 578-581.

[36] E. Macii, M. Pedram, and F. Somenzi, "High-Level Power Modeling, Estimation, and Optimization," *IEEE Trans. CAD of Integrated Circuits and Systems*, vol. 17, no. 11, Nov. 1998, pp.1061-1079.

[37] S. Ravi, A. Raghunathan, and S. Chakradar, "Efficient RTL Power Estimation for Large Designs," *Proc. Int'l Conf. VLSI Design*, IEEE Press, 2003 pp.431-439.

[38] V. Tiwari, S. Malik, and A. Wolfe, "Power Analysis of Embedded Software: A First Step toward Software Power Minimization," *IEEE Trans. VLSI Systems*, vol. 2, no. 4, Dec. 1994, pp.437-445.

[39] C.T. Hsieh, M. Pedram, H. Metha, and F. Rastgar, "Profile Driven Program Synthesis for Evaluation of System Power Dissipation", *Proc. Design Automation Conference* (DAC 97), IEEE Press, 1997, pp.576-581.

[40] C. Brandolese, F. Salice, W. Fornaciari, and D. Sciuto "Static Power Modeling of 32-Bit Microprocessors," *IEEE Trans. CAD of Integrated Circuits and Systems*, vol. 21, no. 11, Nov. 2002, pp. 1306-1316.

[41] T. Givargis, F. Vahid, and J.Henkel, "Trace-driven System-level Power Evaluation of System-on-a-chip Peripheral Cores," *Proc. Asia and South Pacific Design Automation Conference* (ASP-DAC 01), IEEE Press, 2001, pp. 306-311.

[42] M. Kamble and K. Ghose, "Energy Efficiency of VLSI Caches: A Comparative Study," *Proc. Int'l Conf. VLSI Design*, IEEE Press, 1997, pp. 261-267.

[43] M.Kamble and K.Ghose, "Analytical Energy Dissipation Models For Low Power Caches," *Proc. Int'l Symp. Low Power Electronics and Design* (ISLPED 97), IEEE Press, Aug. 1997, pp. 143-148.

[44] R.J. Evans and P.D. Franzon, "Energy Consumption Modeling and Optimization for SRAMs," *IEEE J. Solid State Circuits*, vol. 30, no. 5, May 1995, pp.571-579.

[45] J. Hezavei, N. Vijaykrishnan, and M.J. Irwin, "A Comparative Study of Power Efficient SRAM Designs," *Proc. 10th Great Lakes Symp. VLSI*, IEEE Press, 2000, pp. 117-122.

[46] K. Itoh, K. Sasaki, and Y. Nakagome, "Trends in Low-Power RAM Circuit Technologies," *IEEE Proceedings*, vol. 83, no. 4, Apr. 1995, pp. 524-543.

[47] K. Itoh, "Trends in Low-Voltage Embedded-RAM Technology," *Proc. Int'l Conf. Microelectronics* (MIEL 02), IEEE Press, 2002, vol.2, pp. 497-501.

[48] W. Fornaciari, D. Sciuto, and C. Silvano, "Power Estimation for Architectural Exploration of HW/SW Communication on System-level Buses," *Proc. Int'l Workshop Hardware/Software Codesign*, IEEE Press, 1999, pp. 152-156.

[49] M.R. Stan and W.P. Burleson, "Bus Invert coding for low power I/O," *IEEE Trans. VLSI Systems*, vol. 3, no. 1, Mar. 1995, pp. 49-58.

[50] L. Benini, R. Hodgson, and P. Siegel, "System-level Power Estimation and Optimization," *Proc. Int'l Symp. Low Power Electronics and Design* (ISLPED 98), IEEE Press, 1998, pp.173-178.

[51] J. Henkel and Y. Li, "Avalanche: An Environment for Design Space Exploration and Optimization of Low-Power Embedded Systems," *IEEE Trans. VLSI Systems*, vol.10, no. 4, Aug. 2002, pp. 454-468.

[52] T. Givargis, F. Vahid, and J. Henkel, "Evaluating Power Consumption of Parameterized Cache and Bus Architectures in System-on-a-Chip Designs," *IEEE Trans. VLSI Systems*, vol. 9, no. 4, Aug. 2001, pp. 500-508.

[53] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimization," *Proc. 27th Int'l Symp. Computer Architecure*, IEEE Press, 2000, pp. 83-94.

[54] D. Burger and T.M. Austin, *The SimpleScalar tool set, version 2.0*, tech. report 1342, Computer Sciences Dept., Univ. Wisconsin, Madison, 1997.

[55] M.D. Hill et al., *WARTS: Wisconsin Architectural Research Tool Set*, Computer Sciences Dept., Univ. Wisconsin, Madison, 1999.

[56] T.Givargis and F.Vahid, "Platune: A Tuning Framework for System-on-a-Chip Platforms," *IEEE Trans. CAD of Integrated Circuits and Systems*, vol.21, no.11, Nov. 2002, pp.1317-1327.

[57] http://www.chipvision.com/

[58] L. Kruse, *Estimating and Optimizing Power Consumption of Integrated Macro Blocks at the Behavioral Level*, doctoral dissertation, Univ. Oldenburg, Oldenburg, Germany, 2001.

[59] E. Schmidt, *Power Modeling of Embedded Memories*, doctoral dissertation, Univ. Oldenburg, Oldenburg, Germany, 2003.

[60] C. Talarico, J.W. Rozenblit, and A. Stritter, "A Hybrid Approach for Performance Assessment in Embedded Systems Design," to be published in *Proc. Conf. Conceptual Modeling and Simulation* (CSM-I3M 2004), SCS Publications, San Diego, CA, 2004.

[61] F. Vahid and T. Givargis, *Embedded System Design A Unified Hardware/Software Intruduction*, John Wiley & Sons, 2002.

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

[63] M. Sgroi, L. Lavagno, and A. Sangiovanni-Vincentelli, "Formal models for Embedded System Design," *IEEE Design & Test of Computers*, Apr. 2000, pp. 2-15.

[64] C. Talarico, J.W. Rozenblit, A. Gupta, and E. Peter, "Performance Analysis of Embedded Systems with SystemC," to be published in *Proc. Int'l Conf.*

Computing, Communications and Control Technologies (CCCT 2004), IIIS Press, Orlando, FL, 2004 (invited paper).

[65] J. Sztipanovits and G. Karsai, "Model-integrated computing," *IEEE Computer*, Apr. 1997, pp. 110-112.

[66] E.A. Lee, S. Neuendorffer, and M.J. Wirthlin, "Actor-oriented Design of Embedded Hardware and Software Systems," *J. Circuits, Systems, and Computers*, vol. 12, no. 3, World Scientific Publishing, River Edge, NJ, June 2003, pp. 231-260.

[67] J-H. Chern, J. Huang, L. Arledge, P-C. Li, and P. Yang, "Multilevel Metal capacitance Models for CAD Design Synthesis Systems", *IEEE Electron Device Letters*, vol. 13, no. 1, Jan. 1992, pp. 32-34.

[68] J. Bhasker, *A SystemC Primer*, Star galaxy Pub., Allentown, PA, 2002.

[69] Open SystemC Initiative, *Functional Specification for SystemC 2.0*, 2002, http://www.systemc.org

[70] Open SystemC Initiative, *SystemC Version 2.0 User's Guide*, 2002, http://www.systemc.org

[71] T. Groetker, S. Liao, G. Martin, and S. Swan, *System Design with SystemC*, Kluwer Academic, 2002.

[72] C. Talarico, J.W. Rozenblit, V. Malhotra, and A. Stritter, "Early Design Assessment in Embedded Systems Engineering: a Power Estimation Application", submitted for publication to *IEEE Computer.*

[73] Synopsys, Design Compiler Reference Manual, June 2003.

[74] Synopsys, Power Compiler User Guide, June 2003.

[75] Synopsys, Power Compiler Reference Manual, June 2003.

[76] A.K. Sharma, *Advanced Semiconductor Memories: Architectures, Designs, and Applications*, IEEE Press, 2003.

[77] C. Talarico, V. Malhotra, J.C. Vial, and M. Hage-Hassan, "System Level Power Estimation of System-on-a-Chip SRAMs," to be published in *Proc. Int'l Conf. Computing, Communications and Control Technologies* (CCCT 2004), IIIS Press, Orlando, FL, 2004 (invited paper).

[78] C. Talarico, V. Malhotra, J.C. Vial, and M. Hage-Hassan, "High Level Power Modeling of Embedded SRAMs," submitted for publication to *J. Embedded Computing*, Cambridge Int'l Science, Cambridge, MA.

[79] M. Dumpos and C. Zopounidis, *Multicriteria Decision Aid Classification Methods*, Kluwer Academic, 2002.

[80] P. Garg, A. Gupta, and J.W. Rozenblit, "Performance Analysis of Embedded Systems in Virtual Component Co-design Environment", *Proc. 11th Ann. Int'l Conf. and Workshop Engineering of Computer Based Systems* (ECBS 04), IEEE Press, 2004.

[81] E. Triantaphyllou, *Multi-criteria Decision Making Methods: A Comparative Study*, Kluwer Academic, 2000.

[82] T.L. Saaty, *Models, Methods, concepts and Applications of the Analytic Hierarchy Process*, Kluwer Academic, 2001.

[83] R. Sreeramaneni, *Energy Profiler for Hardware/Software Co-Design*, master thesis, Dept. Electrical and Computer Engineering, Univ. Arizona, Tucson, 2003.

[84] T. Givargis, F. Vahid, and J. Henkel, "Trace-driven System-level Power Evaluation of System-on-a-chip Peripheral Cores," *Proc. Asia South Pacific Design Automation Conf.* (ASP-DAC 01), IEEE Press, 2001, pp.306-311.

[85] F. Vahid and T. Givargis, "Incorporating Cores into System-Level," *Int'l. Symp. System Synthesis* (ISSS98), IEEE Press, 1998, pp.43-48.

[86] T. Simunic, L. Benini, and G. De Micheli, "Cycle-Accurate Simulation of Energy Consumption in Embedded Systems," *Proc. 36th Ann. Design Automation Conference* (DAC 99), IEEE Press, 1999, pp. 867-872.

[87] S.R. Powell and P.M. Chau, "A Model for Estimating Power Dissipation in a Class of DSP VLSI Chips," *IEEE Trans. Circuits and Systems*, vol. 38, no. 6, June 1991, pp. 646-650.

[88] J. Henkel and R. Ernst, "High-Level Estimation Techniques for usage in Hardware/Software Co-Design," *Proc. Asia South Pacific Design Automation Conf.* (ASP-DAC 98), IEEE Press, 1998, pp.353-360.

[89] K.M. Büyükşahin and F.N. Najm, "High-Level Power Estimation with Interconnect Effects," *Proc. Int'l Symp. Low Power Electronics and Design* (ISLPED 00), IEEE Press, 2000.

[90] S. Schulz, J.W. Rozenblit, M. Mrva, and K. Buchenrieder, "Model-Based Codesign," *IEEE Computer*, Aug. 1998, pp. 60-67.

[91] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. Trakhtenbrot, "STATEMATE: A working Environment for the Development of Complex Reactive Systems," *IEEE Trans. Software Engineering*, vol. 16, no. 4, Apr. 1990, pp. 403-414.

[92] A. Sangiovanni-Vincentelli and G. Martin, "Platform-Based Design and Software Design Methodology for Embedded Systems," *IEEE Design & Test of Computers*, Dec. 2001, pp. 23-33.

[93] A. Raghunathan, S. Dey, and N.K. Jha, "High-Level Macro-Modelin and estimation Techniques for Switching Activity and Power Consumption," *IEEE Trans. VLSI Systems*, vol. 11, no. 4, Aug. 2003, pp. 538-557.

[94] S. Gupta and F.N. Najm, "Energy and Peak-Current Per-Cycle Estimation at RTL," *IEEE Trans. VLSI Systems*, vol. 11, no. 4, Aug. 2003, pp. 525-537.

[95] M. Barocci, L. Benini, A. Bagliolo, B. Riccó, and G. De Micheli, "Lookp Table Power Macro-models for Behavioral Library Components," *Proc. Alessandro Volta Memorial Workshop Low Power Design*, IEEE Press, 1999.

[96] T. Givargis, F. Vahid, and J. Henkel, "Fast Cache and Bus Power Estimation for Parameterized System-on-a-Chip Design," *Proc. Conf. Design Automation and Test in Europe* (DATE 00), IEEE Press, 2000.

[97] T. Givargis, J. Henkel, and F. Vahid, "Interface and Cache Power Exploration for Core-Based Embedded System Design," *Int'l Conf. Computer Aided Design* (ICCAD 99), IEEE Press, 1999, pp. 270-273.

[98] T. Givargis, F. Vahid, and J. Henkel, "A Hybrid Approach for Core-Based System-Level Power modeling," *Proc. Asia South Pacific Design Automation Conf.* (ASP-DAC 00), IEEE Press, 2000, pp. 141-145.

[99] J. Henkel, "A Low Power Hardware/Software Partitioning Approach for Core-Based Embedded Systems," *Proc. 36th Ann. Design Automation Conf.* (DAC 99), IEEE Press, 1999, pp. 122-127.

[100] F. Vahid and T. Givargis, "Platform Tuning for Embedded Systems Design," *IEEE Computer*, March 2001, pp. 112-114.

[101] S. Edwards, L. Lavagno, E.A. Lee, and A. Sangiovanni-Vincentelli, "Design of Embedded Systems: Formal Models, Validation, and Synthesis," *Proc. IEEE*, vol. 85, no. 3, March 1997, pp. 366-390.

[102] A. Kaladave and E.A. Lee, "A Hardware-Software Codesign Methodology for DSP Applications," *IEEE Design and Test of Computers*, Sept. 1993, pp. 16-28.

[103] T. Simunic, J.W. Rozenblit, and J.R. Brews, "VLSI Interconnect Design Automation Using Quantitative and Symbolic Techniques," *IEEE Trans. Components, Packaging and Manufacturing Technology*, vol. 19, no. 4, Nov. 1996, pp. 803-812.

[104] R.J. Gupta and S.Y. Liao, "Using a Programming language for Digital System Design," *IEEE Design and Test of Computers*, June 1997, pp. 72-80.

[105] G. De Micheli and R.K. Gupta, "Hardware/Software Co-Design," *Proc. IEEE*, vol. 85, no. 3, March 1997, pp. 349-365.

[106] W. Wolf, "A Decade of Hardware/Software Codesign," IEEE Computer, Apr. 2003, pp. 38-43.

[107] J.W. Rozenblit, "Experimental Frame Specification Methodology for Hierarchical Simulation Modeling," *Int'l. J. General Systems*, vol. 19, no. 2, Feb. 1991 pp. 317–335.

[108] G. Stitt and F. Vahid, "Energy Advantages of Microprocessor Platforms with On-Chip Configurable Logic," *IEEE Design and Test of Computers*, Dec. 2002, pp. 36-43.

[109] W. Strunk Jr. and E.B. White, *The Elements of Style,* 4th ed., Longman, 2000.