UNIVERSITY OF CALIFORNIA,
IRVINE

Automated Power-Aware System-Level Design

with the MAVO Framework

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Electrical Engineering and Computer Engineering

by

Yasaman Samei

Dissertation Committee:
Professor Rainer Dömer, Chair
Professor Elaheh Bozorgzadeh
Professor Fadi Kurdahi

2014

DEDICATION

to my loved ones

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

It gives me great pleasure to take this opportunity to acknowledge all the people who supported me in the last four years of my PhD.

First and for most I would like to express my gratitude to my supervisor Prof. Rainer Dömer, whose insights, expertise, understanding and above all patience added considerable to my graduate experience. I appreciate his great research skills, vision, ethics and joy for new ideas. I strongly believe the LECS enjoyable and great research environment is all thanks to Prof. Rainer Dömer.

I would like to thank Prof. Elaheh Bozorgzadeh and Prof. Fadi Kurdahi for intellectual contribution to my development as a researcher, and of course taking time out of their busy schedules to serve as my committee member.

I most also acknowledge the CECS members and my lab mates, for the great discussions and motivation. Many thanks to Dr. Weiwei Chen, Dr. Xu Han, Guantao Liu and specially Che-Wei Chang for sharing their provision and thoughtful suggestions.

My appreciation also goes to lovely Susan Staebell and Loretta Waltemeyer, the EECS department staff who always helped and supported me and all EECS students.

I would like to thanks all my friends who have been a great source of inspiration and support through all these bumpy year of PhD life. Dear Sara, Amin, Shaahin, Bahare, and Farshad, I always felt so honored and happy to have you in my life. My special thanks to lovely Sara and Amin, who are a great source of fun and emotional support, Shaahin for his motivations and guidance and Bahare and Farshad for their helps and sharing their excellence advices. I also like to thank my friends in EECS department Ashkan, Pouria and Mahshid for their fruitful help and valuable suggestions.

My most heartfelt thanks to Gita and Reza, my parents who always encouraged me to be independent thinker and having confidence in my abilities to go after new things that inspires me. Thank you for your constant support through the ups and down of my life. Great thanks to my brother Mohamad who supported me and believed in me throughout my whole life.

Last by not least I would like to thank Sam. Dear Dr. Sam Pournazeri who has been by my side from the first day of my PhD to this day. His beautiful character, enthusiasm for life and success is a great source of inspiration and I cannot adequately express how thankful I am. Thanks for everything that helped me get to this day.

CURRICULUM VITAE

Yasaman Samei

Education

Doctor of Philosophy, Electrical and Computer Engineering **2014**
Department of Electrical Engineering and Computer Science
University of California, Irvine

Master of Science, Computer Science **2010**
Department of Computer Science and Engineering
University of California, Riverside

Bachelor of Engineering, Computer Science and Engineering **2008**
Department of Electrical Engineering and Computer Engineering
Shahid Beheshty University

Research Experience

University of California, Irvine Irvine, CA
Graduate Student Researcher 2010-2014
Center for Embedded Computer Systems
Department of Electrical Engineering and Computer Science

University of California, Riverside Riverside, CA
Graduate Student Researcher 2009-2010
Department of Computer Science and Engineering

Shahid Beheshti University Tehran, Iran
Undergraduate Student Researcher 2007-2008
Department of Computer Engineering

Publications

- Yasaman Samei and Rainer Dömer. Automated Estimation of Power Consumption for Rapid System Level Design. In *International Performance Computing and Communications Conference (IPCCC)*. IEEE, 2014
- Yasaman Samei and Rainer Dömer. PowerMonitor: A Versatile API for Automated Power-Aware ESL Design. In *Forum on Specification & Design Languages (FDL)*. IEEE, 2014

- Yasaman Samei and Rainer Doemer. MAVO: An Automated Framework for ESL Design Monitor, Analyze, Visualize and Optimize. 2014
- Takuya Azumi, Yasaman Samei Syahkal, Yuko Hara-Azumi, Hiroshi Oyama, and Rainer Dömer. TECSCE: HW/SW Codesign Framework for Data Parallelism Based on Software Component. In *Embedded Systems: Design, Analysis and Verification*, pages 1–13. Springer, 2013
- Xu Han, Yasaman Samei, and Rainer Doemer. System-level modeling and refinement of a canny edge detector. *Center for Embedded Computer Systems*, 2012
- Deepak Mishra, Yasaman Samei, Nga Dang, R Domer, and Elaheh Bozorgzadeh. Multilayer configuration exploration of MPSoCs for streaming applications. In *Electronic System Level Synthesis Conference (ESLsyn), 2012*, pages 38–43. IEEE, 2012
- Hadi S Aghdasi, Maghsoud Abbaspour, Mohsen Ebrahimi Moghadam, and Yasaman Samei. An energy-efficient and high-quality video transmission architecture in wireless video-based sensor networks. *Sensors*, 8(8):4529–4559, 2008

Teaching Experience

University of California, Irvine
Teaching Assistant, Department of Electrical Engineering and Computer Science

- Software Engineering Project in C Language	Winter 2014
- Advanced C Programming	Fall 2013
- Digital Design	Summer 2012, 2013, Spring 2013, 2014
- Digital Design Laboratory	Winter 2011, 2012, 2013, Summer 2011
- Computational Method in Computer Engineering	Fall 2011, 2012
- Computer Graphics	Spring 2011

University of California, Riverside
Teaching Assistant, Department of Computer Science and Engineering

- Operating Systems	Winter 2009
- Introduction to Programming	Spring 2009

Professional Experience

Microchip	Irvine, CA
Validation Test Engineer Intern	2014
ALTERA	San Jose, CA
SoC FPGA Product Planning Intern	2012

Honor and Awards

- Travel grant, International Performance Computing and Communications Conference (IPCCC) 2014

- Achieved the second place in the ACM International Collegiate Programming Contest, 2005 (Qualified for World Final ACM International Collegiate Programming Contest)
- Achieved the second place in the ACM International Collegiate Programming Contest, 2006

ABSTRACT OF THE DISSERTATION

Automated Power-Aware System-Level Design

with the MAVO Framework

By

Yasaman Samei

Doctor of Philosophy in Electrical Engineering and Computer Engineering

University of California, Irvine, 2014

Professor Rainer Dömer, Chair

For the past few decades, semiconductor capabilities have been improving as Moore's law predicted. Transistor size has been shrinking and technology size will be less than 20nm in the near future. These improvements enable designers to come up with more complex systems. However, this has made power dissipation a major design obstacle. Conventionally, power consumption is considered in the later stages of the design process, like the architecture level, Register Transfer Level (RTL), gate level, and physical level, where detailed information about the design is available. Although there are many power-aware design tools at these lower levels, the simulation and evaluation time are high and often beyond the time-to-market requirements. To tackle the long simulation time as well as avoiding time consuming design modifications at lower levels, designers are raising the level of abstraction to the system level.

Over the last decade, research in Electronic System Level (ESL) design has resulted in significant advances in addressing the rising design complexity and meeting the required performance constraints. Now a major concern of system-level design is power dissipation in System-on-Chip (SoC) which not only affects battery lifetime but also thermal aspects and reliability of the end product. Although power aware design is crucial in ESL design,

System Level Description Languages (SLDL) are not supporting this feature natively.

Towards power-aware ESL design, in this dissertation we present *MAVO*, an automated framework to *Monitor, Analyze, Visualize* and *Optimize* both power and performance at the early stages of the design process. The proposed framework supports waveform-based power estimation and optimization for rapid system-level design. MAVO is adapted for automated SoC design and it is integrated to System-on-Chip Environment, a prototype ESL design tool for rapid power-aware design.

We perform different experiments to evaluate the accuracy and fidelity of the framework, including JPEG image encoder, MP3 audio decoder and H.264 video decoder and encoder. Experimental results show that our developed framework can achieve a high degree of fidelity while providing significant speedup.

We also examined MAVO for applying different power optimization mechanisms on a Canny edge detector application. Our studies show large potential for design modification toward power efficient design models at system-level. Additionally we applied MAVO along with static analyzer tool in order to capture power and performance trade-offs and apply power optimization techniques automatically.

Overall, our work provides an advanced power estimation infrastructure for power- and performance-aware system model development. It can significantly help embedded system designers to build low-power and reliable products in shorter time frame.

Chapter 1

Introduction

1.1 Embedded Systems

Embedded systems are getting more and more attention every day through presenting smartness in all sorts of devices from cellphones to household and office equipment. The embedded computer systems are formally defined as *special purpose information systems that are embedded in larger systems to provide operations without human intervention* [59], and informally defined as *a collection of programmable parts surrounded by Application-Specific Integrated Circuits (ASICs) and other standard components, that interact continuously with an environment through sensors and actuators* [9].

Although embedded systems have emerged later in the field, they are dramatically moving the semiconductor and electronics industry in new directions supporting growth and continued innovation. In semiconductor industry where the technology size is shrinking as Moore's law predicted and multi-tasking devices are the new trend to achieve higher performance, embedded systems are getting more and more attention. Embedded systems have found a widespread application in both hardware and software market across various industries such

as automotive, healthcare, telecommunications, military and aerospace. These extensive markets for embedded systems demands for more design tasks, where these tasks need to be performed quickly, efficiently, and reliably.

The above mentioned requirements have made embedded systems design a challenging process and with increasing complexity of these systems, embedded systems design is the focus of many research efforts.

In this chapter we look into embedded system design methodology, different design abstraction levels, low power design and related works in this area.

1.2 Embedded System Design Over Time

The complexity growth in embedded systems does not allow designers to follow the old design methodologies any more (The aggregation of models, components, guidelines and tools is called design methodology [30]). During the last few decades three main methodologies have been introduced in embedded systems design field [32]:

- **Capture and Simulate Methodology**

  Embedded system design flow lacks tools and formal approaches for model generation, simulation, synthesis and verification. Embedded systems are conventionally designed in an ad hoc manner that is highly depend on previous products and designer's experience. Originally the embedded systems manufacturers begin the design process based on the specification of the design for hardware and software separately. The main design specifications and requirements were implemented at software side. Then hardware engineers took over and develop the hardware side. However, matching sides, design testing and verification was not possible until hardware side were complete. Therefore the process was long and debugging and design modification were postponed to the end of design-cycle.

- **Describe and Synthesis Methodology**

  To improve the design process, synthesis tools were introduced, which allow the designers to develop their design behavior in forms of Boolean equations and Finite State Machines (FSM) and automatically generate the corresponding structural model as a gate netlist. However the order of complexity and size of embedded system was not compatible with required equivalency check between behavioral and structural models in this approach. The abstraction of gate level to RTL did not help to resolve the design gap as well. As a result in early 2000s system-level design concept emerged to tackle this deficiency.

- **Specify, Explore and Refine Methodology**

  In this methodology initially the design model is specified and it goes through different exploration and refinement steps to achieve the correctness, performance and timing criteria. During this era system-level design was proposed to simplify the design flow. In order to strengthen the system-level design through refinement, an explicit semantic was needed in embedded system design languages which was absent in High Level (HL) and Hardware Description Languages (HDL).

  The *Specify, Explore and Refine* approach is also applied in this work and will be described in details in next section.

1.3 Levels of Abstraction

The different level of abstractions are presented in Figure 1.1. As it is shown the higher the level of abstraction goes the number of components available at each level becomes less with an orders of magnitude or more. Therefore system modeling, design and verification are all accelerated in correlate with the complexity of the models in different abstraction levels. However, since the number of the components are reduced due to abstraction, less

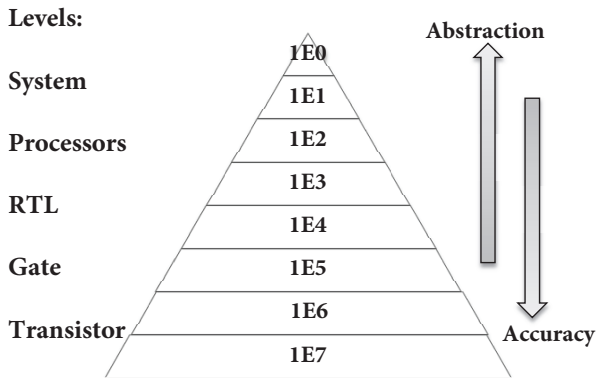


Figure 1.1: Levels of abstraction in SoC design [35]

information from the design and design libraries are available. Thus, in higher levels design evaluation and constraints verification accuracy degrade to some extent accordingly.

1.4 System Level Design

The current semiconductor technology allows the designers to integrate hardware accelerators, multiple processors, communication hierarchy and IO devices and drivers into a single chip so called the Multi-processor System-on-chip (MPSoC). This strong suit is a great potential for developing advanced embedded systems, however the design process can be extremely sophisticated and requires design automation tools.

Not only different high volume of design specification and implementation concerns are needed but also new design problems such as high power dissipation and reliability issues are added to designers concerns list. Figure 1.2. shows a design process deficiencies which is caused by the above mentioned capabilities. As it is shown by International Technology Roadmap for Semiconductor (ITRS) [25] the embedded system design process is suffering from the hardware and software design gap.

As it is shown the software productivity is far beyond the rapid hardware and technology improvements. The current need for software gets doubled every 10 month where the current

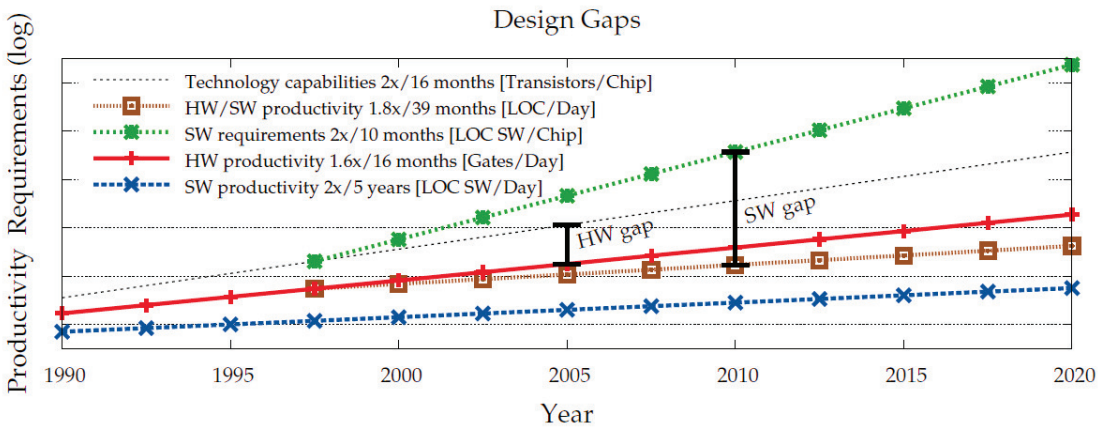


Figure 1.2: Hardware and software design gaps versus time (Source[60])

rate is 5 years. In semiconductor industry where all the focus was on improving technology, now software productivity is the bottle neck of the design process. The current speed of software development for SoCs cannot utilize the complexity of the available HW.

To simplify the specification, verification and implementation of system in both hardware and software, accelerating the design space exploration process as well as narrowing down the productivity gap, a new level of abstraction called *System-Level* is introduced. In system-level design, specifications are implemented in SLDLs as executable models which can be tested and verified. On the other hand, the structural models are described in higher level of abstraction with computation and communication apart from each other.

1.5 System-Level Design Methodologies

Although system-level design proposed a better solution for embedded system design, the design flow is still complex and time-consuming. To alleviate the problem Computer Aided Design (CAD) tools are improving toward automating the design flow and lightening the manual efforts which eliminates the human error as well.

There are three main techniques in design methodology; the *bottom-up*, *top-down* and *meet-*

in-the-middle. In first method the design process start from lowest level of the design, the transistor level, and design gets abstracted level by level through generating libraries, with details on layout and floor plan. In *top-down* approach the exact opposite direction is taken. Design flow starts from highest level of abstraction, system-level. At this level functional description is prepared and goes through design decisions and refinements step to enrich with more details. In the third method *meet-in-the-middle*, as it is implied by its name design process start from both directions, highest and lowest level of abstraction simultaneously, and then merged in one of the mid-levels like RTL or Gate level.

Each of these methods has their own advantage and disadvantages. In *bottom-up* approach although the design constraints can be evaluated with highest level of accuracy, in order to obtain the best design option every parameter needs to be altered in all the different design models within different abstraction levels. In *top-down* approach, it is not possible to evaluate the design constraints with high level of accuracy initially, since system-level design model does not cover the detail information of the design. On the other hand design customization and verification is easy and scalable. The *meet-in-the-middle* has both of the mentioned disadvantages in other two methods, however, it is more accurate in compare to *top-down* approach and requires fewer layouts than *bottom-up*.

Apart from these three methods, platform methodology, system methodology and FPGA methodology are also evaluated in [30] which are popular in companies with in house design tools.

1.6 Electronic System Level Design Process

In this work we are using a top-down approach for ESL design process. In this process we are relying on Y-chart [31] for showing design flow and X-chart [36] for synthesis at each design abstraction level.

1.6.1 Y-Chart

The Y-chart is a well-known representation of different perspectives in the embedded system design. It combines the presentation of the five hierarchy levels and three design description domains. The axes of the chart are the domains and circles are the abstraction levels.

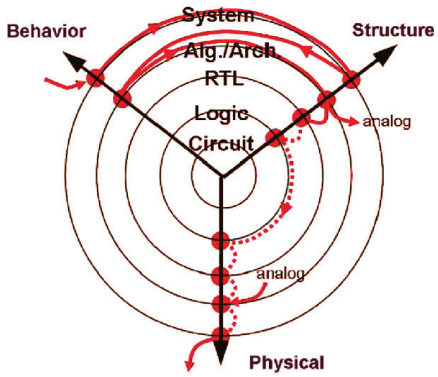


Figure 1.3: Top-down system-level design in the Y-chart [31] (Source [83])

In the Y-chart the domains are behavioral, structural and physical. The behavioral domain describes the temporal and functional behavior of the system. The structural domain is structured from subsystems which different subsystems and their interconnections are listed for each level of abstraction. Finally the physical domain represents the geometric properties of the system and its subsystems. At this domain models have detail information on size, area and placement of the system.

The abstraction levels and their accuracy and relations are presented in Section 1.3. In Figure 1.3, a top-down methodology flow is marked by red in the Y-chart. In this approach the design methodology begins at system-level as the highest abstraction level to convert the system-level description of the design to component netlist at each abstraction level. In this flow design models are equipped with more details step by step in the design flow. In this

work we are using the top-down approach as well.

1.6.2 X-chart

During the synthesis process a specification is converted to an implementation [36]. Figure 1.4, demonstrates the synthesis flow in our work. Here synthesis starts with defining the

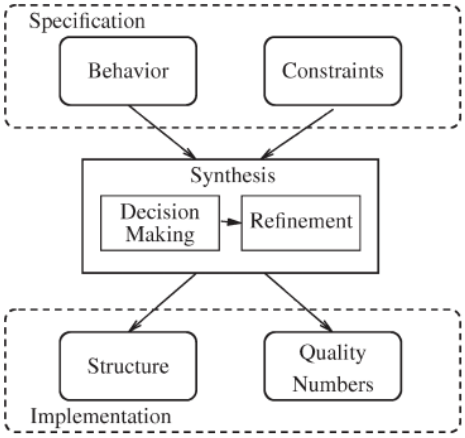


Figure 1.4: Synthesis flow in the X-Chart (Source [36])

specifying model behavior and design constraints. The behavioral model is the system functionality, which can be generated based on different models of computations and in different programming languages. According to model abstraction level, SLDLs, e.g., SpecC [34], SystemC [2], or HDLs e.g., Verilog [3] or VHDL [4] can be used.

Design constrains need to be evaluated from different aspect. For instance, the available resources, such as processors, number of them, different communication schemes or memory hierarchies. The design constraints can also be performance, power or area limitations. All these criterion should be defined for synthesis process.

Next base on the designer decisions and through one or multiple refinements steps, the specification model will get converted to an implementation. The implementation model composed

of a structural model and design quality evaluation numbers. The structural model reflects the behavioral model and all the architectural decisions that have been applied, such as PE allocations and mappings, scheduling and communication schemes.

1.6.3 Refinement-Based ESL Design Flow

In our method design process starts with high level description of the design which is usually specified in one of the SLDLs, then the model get validated through simulation for correctness and finally the model is refined through a CAD tool. These three steps are called: *specification, validation and refinement*. This process can be iterated multiple times for design optimization and further changes in the design model. An overview of this approach is presented in Figure 1.5.

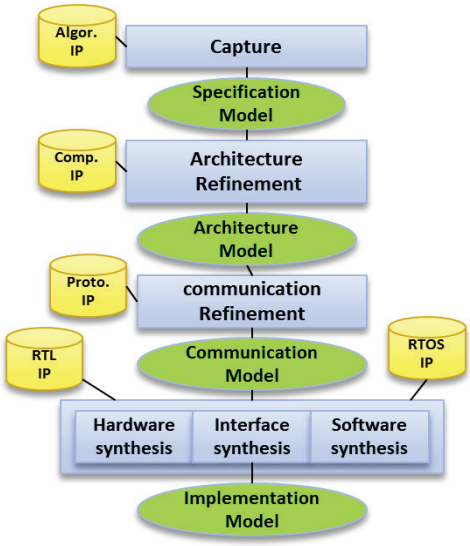


Figure 1.5: Refinement-based ESL design flow [26]

1.7 Low Power Design and Electronic System Level

Although the ESL design tools and models have alleviated the embedded system design problems and reduced the design-cycle time enormously, due to complexity of the systems and different kinds of constraints, such as physical constraints (size, weight etc.), functional constraints (performance, throughput, power consumption, etc.), environment constraints (humidity, temperature, etc.) and economic constraints (cost, time to market etc.) it is still a complicated process.

One of the major design concerns which is getting more and more critical due to reduction in technology size is power. The advances in CMOS technology can accommodate more computing elements on chip and more applications on hardware. The integration of all these elements has resulted in high power dissipation in embedded systems. The silicon technology advance has produced more reliability, variability and leakage power challenges [65]. The ITRS [70] is predicting that the power consumption of stationary devices increase by a factor of two and the leakage and dynamic power consumption will be equal for both logic and memory parts.

Conventionally power reduction techniques and power optimization solutions are applied and inserted in lower level of the design such as RTL, Gate Level and Transistor Level, since in lower level more information are available and therefore power analysis is more accurate. On the contrary, in ESL power analysis, accuracy is degraded due to abstraction, since less design information is available or partially known, such as design characterization, switching activity, and application test-bench of hardware resources.

On the other hand the power saving opportunity is reduced from about 10X-50X at system-level to 10%-50% at transistor level [66] as it is shown in Figure 1.6. Therefore, postponing the power analysis to lower level of the design will disregard the great opportunity of power optimization at ESL.

Another encouraging point about initiating power evaluation during ESL design is the expectation of accuracy at ESL. Due to the fact that at ESL design the main concern is to make the correct design decisions, the concept of relative accuracy or in another word fidelity can replace the absolute accuracy. Fidelity is used as a quality metric for estimator evaluation. It is defined as the percent of correct estimations for pairs of design implementation [33]. The accuracy-speed trade-off along with fidelity requirements in ESL design have been the motivation of this thesis. Our main goal in this thesis is to improve the effectiveness of power evaluation and provide an automated framework for power optimization at ESL.

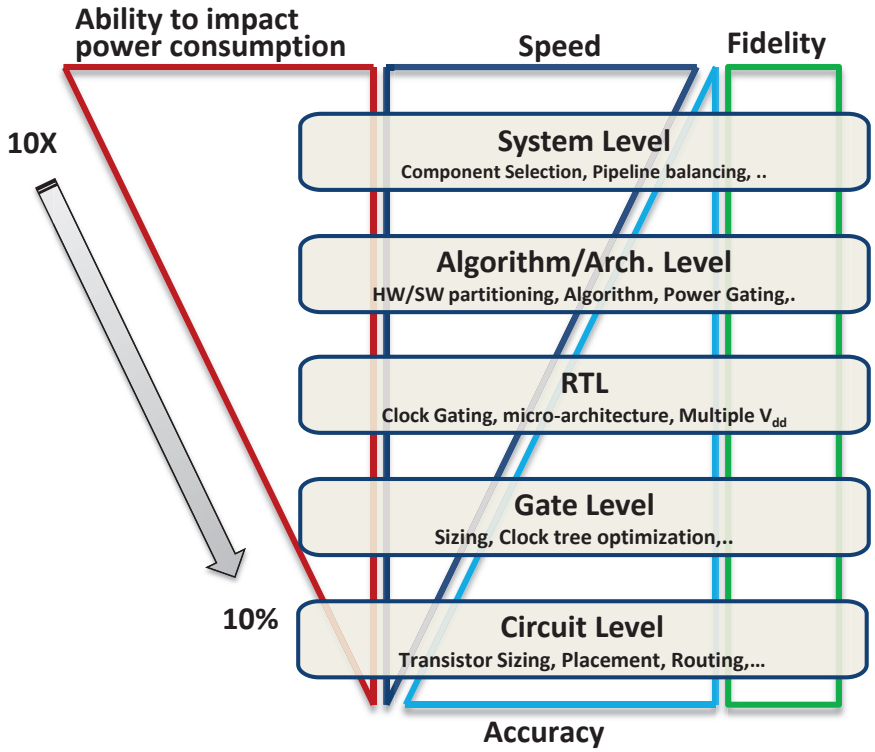


Figure 1.6: Ability to impact and quantify power dissipation over the different design abstraction layers [83]

The speed, accuracy, and fidelity are the main evaluation metrics for system-level estimators and we examined MAVO framework against them in Chapter 3. Among these metrics we can improve the speed and preserve the fidelity of our system-level power estimator, but the accuracy is expected to degrade in compare to lower levels power estimators, Figure 1.6.

1.7.1 Power

The power dissipation in embedded systems has different sources, all the resources activity and interactions, as well as environmental factors have an impact on power consumption. The power dissipation is usually grouped in: *Dynamic Power*, *Static Power* and *Short-Circuit Power*.

- **Dynamic Power**

  Dynamic power dissipation or active power is due to activity and switching frequency. Charging and discharging the capacitors are the main reasons for dynamic power dissipation. The dynamic power consumption can be formulated as [65]:

$$P_{Dynamic} = C_{switched} \times V_{DD}^2 \times f \tag{1.1}$$

  Where $C_{switched}$ is the switched capacitance, V_{DD} is the power supply voltage and f is the activity frequency. The unit of the resulted metric is *Watt*(*Joule/sec*).
  Dynamic power is highly dependent on application and applied architecture and it is the dominant source of power consumption in embedded systems [7]. At system-level, there is a huge potentials for reducing power dissipation of hardware design through altering capacitance and activity profile of the design via hardware changes.

- **Static Power**

  Static power is not depend on device activity, at it is dissipated while device is in non-conducting state. Static power can be expressed as:

$$P_{Static} = V_{DD} \times I_{leakage} \tag{1.2}$$

  Where V_{DD} is the power supply voltage and $I_{leakage}$ is the leaked current. The major factors in static power dissipation are sub-threshold leakage, junction leakage and gate

leakage in transistors. The fact that transistors are working as imperfect switches is the main cause of static power dissipation. With traditional technologies the contribution of dynamic power was about 80% to 90%, however for deep submicron (below 65nm) wiring the capacitor is the major power dissipation source [38] and it is increasing exponentially [7].

- **Short-Circuit Power**

  Short-circuit power is power dissipation due to stacked effect when P and N devices in a CMOS logic gate are in ON state simultaneously [7]. The short-circuit power can be minimized by matching the rise and fall times of the input and output signals in transistors [65]. However the power dissipation due to short circuit can be addresses more effectively in lower levels of the design where more design details are available.

Over all the power dissipation in embedded system hardware design can be summarized as [65]:

$$Power = Energy\ per\ Transition \times Transition\ Rate + StaticPower \tag{1.3}$$

We relied on this formula for power estimation in MAVO framework.

1.8 Dissertation Goals

The CMOS technology is continuously scaling and multiple processors, intellectual properties, and on-chip memories are fully integrated on a small die of a single chip. Therefore, the large-scale MPSoC s requires advance design techniques and efficient design flow. The design flow should support the time-to-market requirements as well as effective design constraints evaluation such as power and performance.

One other major concern that rises along increasing density of transistors is power. The

power evaluation and dissipation is one of the hottest topic, not only for battery power devices but also for other application domain and high performance computing. Key to designing power efficient and reliable system is to initiate power dissipation at highest level of abstraction, the ESL.

This dissertation aims at developing a framework for power estimation and analysis at system-level. The *MAVO* framework will be presented as a simulation-based power estimation tool to *Monitor, Analyze, Visualize* and *Optimize* power behavior of the system-level models . The major contributions of this dissertation are:

- Provide an automated power dissipation *Monitor* to profile the activity of the system-level specification models.
- Develop a power API to support power analysis in ANSI-C based SLDLs and design a model annotator to generate power- and performance-aware models, the *Analyzer*.
- Implement a power *Visualizer* to extract text and graphical report of power dissipation in specification models.
- Improve the observability of power activity in the models for power optimization and design modifications as well as presenting a platform for power-performance design space exploration via integrating the power framework into System-on-Chip design Environment.
- Propose a formal method for capturing power-performance trade-off in different design options rapidly

1.9 Dissertation Overview

This dissertation is organized as follow: In the rest of the chapter 1 the related work and research background will be presented. In chapter 2, the MAVO framework will be presented along with its embedded modules. The JPEG platform is used as a case study in MAVO framework. In Chapter 3, the accuracy and fidelity of the proposed approach is evaluated. A simplified power modeling approach is presented to build a default power models library. The canny application is used for Design space exploration within the MAVO framework in chapter 4. Chapter 5 present different power optimization technique, followed by a case study on Canny application. Chapter 6 describes a static analysis of power-performance at ESL model via ESL model checking. Finally Chapter 7 summarizes the contribution of the work in the dissertation and concludes with a brief discussion on the future work.

1.10 Related Work

There has been large body of work and research efforts on low-power design, power optimization techniques, and power aware Electronic Design Automation (EDA) tools, for design at different levels of abstraction. For power and performance estimation at system level, two common steps within all these studies are: generating power models and tracing model simulation to extract information needed by power models. In the following we look in to related works in these two steps.

1.10.1 Power Modeling

The power modeling and characterization has been studied on all sort of computing devices such as general purpose processors, SoCs and Field Programmable Gate Array(FPGA). The power in SoC is the summation of power consumption in all of the integrated components to the chip, the embedded processors, memory and caches, on-chip Buses, etc.. In order to achieve reasonable power accuracy at system level, all these components are needed to be studied for power modeling and power dissipation influencing factors under different workload. Investigating the power dissipation of these units needs a considerable effort, therefore designers are usually rely on models that are provided by the manufacturers. The commercial CAD tools also have built-in Intellectual Properties (IP) with power model libraries, since extracting them is not possible without knowing the internals of such components and even with black box solutions [50] [56] [79] it is beyond the time-to-market requirements for designers.

The common technique to extract the power models at ESL are *analytical* or *regression-based* [68]. These approaches can be used for all kinds of design components such as processors, custom hardware, interconnections and memories. The power modeling technique can de-

veloped in any of design abstraction levels with different accuracy.

In analytical approach power dissipation is formulated using the structure of the components. Due to complicated nature of investigating the structural details of each components, this approach is usually suitable for well-organized blocks such as memories and caches [57] [76] [85] [46].

The regression method is common for processors and custom hardwares [52], [64]. In this method usually a subset of test cases are exhaustively applied to the system to trace power and performance. For shortening the process the test cases are summarized through grouping different states or operations. In order to improve accuracy, the impact of data dependency and inter-command dependency on power dissipation within each component can be included in the models as well.

The power models result in dissimilar power numbers according to their accuracies and required computational effort for utilizing them. Therefore different power models can be deployed for different design components in simulation, or models can be chosen alternatively for same component depending on the level of desired accuracy [53] [45] [10].

1.10.2 Power Estimation

Power estimation and modeling has been the focus of many research efforts. Although power aware design is crucial in ESL design, SLDLs are not supporting this feature natively.

There are various commercial and noncommercial tools for lower abstraction levels that are accurate but only applicable in low levels where all the the design decisions has been made and design modification are beyond the time to market requirements, such as Spice, Power Compiler [1], PowerPlay [5]. Most of the proposed power estimation methods rely on similar foundations that can be categorized in to three main groups: *cycle accurate*, *instruction based*, and *functional level* based models.

1.10.3 Cycle-Accurate Power Estimators

In cycle accurate methods, such as SimplePower [86], Wattch [14] and McPAT [54] power is analytically quantified by monitoring operations and transactions cycle by cycle and applying power models from the micro-architectural level for each involved unit.

The SimplePower was the first cycle-accurate modeling tool. The proposed model was based on the five-stage pipeline architecture. The collected cycle by cycle information was captured at RTL for each involved functional unit.

Another well-known power model and optimization tool for microprocessor is Wattch. Wattch was proposed at one level higher, at architecture level for modeling dynamic power consumption. In Wattch wide range of information about the architecture such as array structure, content addressable memories, combination logic, wires and clocking are employed for power estimation.

SimplePower and Wattch are both based on super scalar processors and are utilized along with a simulators such as SimpleScalar [15] or GEM [58]. Simulator is responsible for collecting the activity information of the design components for their power models.

The latest power modeling tool set is McPAT (Multicore Power Area and Timing) that is developed by HP Labs. This tool propose models for power, area and timing for multithreaded multicore/manycore architectures. The presented models covers in-order and out-of-order cores, shared on-chip caches, integrated memory controllers, and network on chips for more accurate design evaluation. The major limitation of McPAT is compatibility with other simulators. The Sniper[19] and Multi2Sim [81] are among few simulators that matches the McPAT interface.

In order to estimate the power multiple applications with different workload and data variations are executed on a instruction set simulator. The cycle-accurate simulations of detailed micro-architectural structures are slow. Therefore the long simulation time is the major limitation of power evaluation in this method, albeit the absolute accuracy is reasonably

high.

1.10.4 Instruction Based Power Estimators

An instruction level power analysis tool in [78] was among the first attempt for power evaluation for higher level. In this work the developed power models are in form of a look-up tables and generated based on the power cost of each instruction. Similar instructions were grouped further to simplify the power modeling development process. In this approach the execution of each instruction were captured using an instruction set simulator. Then the total power was calculated by adding up the execution counts and corespondent power numbers.

Powersim [37], an example of an instruction-based power model, presents a C++ class library to be used inside a SystemC design model. Powersim monitors a limited set of SystemC operators, the arithmetic, logic and relational operators, and applies an energy model to monitored operations. Consequently the Powersim is suitable for evaluating the computational dynamic power only. Although Powersim provides the power numbers automatically and without any manual annotation to design model, design components allocation, mapping information along with the power models should be provided to Powersim via configuration files.

1.10.5 Functional Based Power Estimators

The final group is functional level power analysis which is applied in several tools, such as TLM POWER3 [39], and PKtool [82]. In [39], bit level activities are counted in TLM models, while PKtool is presented in the form of a class library for SystemC by means of power estimation and analysis at the system level. Both of these tools help to embed the power

details and abstract power related information, such as Hamming distance of the signals, to the design. However, the process is manual and user-oriented, and therefore neither easy to apply nor scalable.

There are other tools that use the mixture of these approaches. In [69], a hardware/software co-simulation based power estimation and optimization called PETS is proposed. A functional level power analysis approach is used in PETS . PETS uses generic power models with different algorithmic and architectural parameters, while extracting micro architectural activity to tackle the accuracy-speed trade-off. The power modeling selection and design model generation are within an Eclipse based environment with a scripting interface.
COMPLEX [40] is a framework for HW/SW co-design at system levels and allows applying hybrid combination of power models from various works for different design components.

The investigated power estimation approaches in related work section, are compared and summarized in Table 1.1.

Table 1.1: Power estimators comparison

Power Eastimator	Automated	Accuracy	Speed	Power Report
SimplePower	Yes	High	Low	Average Power
Wattch	Yes	High	Low	Average Power
McPat	No	High	Low	Average Power
Powersim	No	Medium	High	Average Power
PKtool	No	Medium	High	Average Power
TLM POWER3	No	Medium	High	Average Power
PETS	No	Medium	Medium	Average Power
COMPLEX	No	Medium	Medium	Power over Time

Chapter 2

System Level Power Estimator

System-level is the starting point for design constraints characterization as well as design space exploration. Therefore design decisions, such as component selections, hardware/software partitioning, communication schemes, number of cores, and power reduction techniques, are ideally all made at the system level. Here, it is critical to make correct decisions. To achieve this goal, a structured ESL tool suite is required to perform assessments.

In this chapter, MAVO, a rapid and automated system-level power estimator, is introduced to monitor the energy consumption of a system-level model by extracting comprehensive power activity of the modules without any manual modifications.

2.1 MAVO Framework

An overview of the design flow using MAVO is presented in Figure 2.1. The main developed modules are a *Monitor* [72], a *PowerAnalyzer* API [73], a power-time model *Annotator*, and an interactive power-performance *Optimizer*.

As it is shown in Figure 2.1, the design process at system level starts with a specification

model, that reflects the functional behavior of the system, without any notion of time nor power. Next, Processing Element (PE) allocations, behavior mappings, scheduling and com-

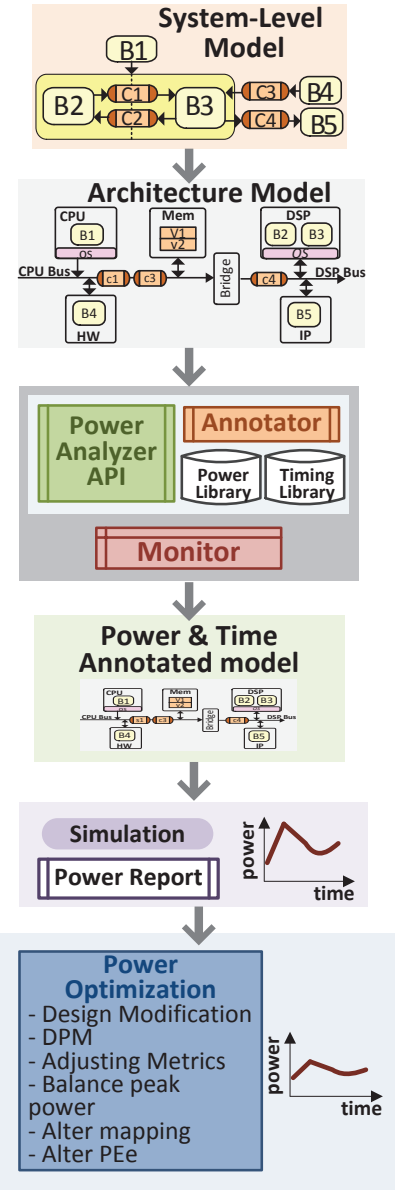


Figure 2.1: Design flow with MAVO framework

munication refinements are performed. All these design decisions generate a large design space which needs to be minimized through elimination, based on design constraints. All invalid design options are pruned from the design space and the best design options go to

power optimization and performance improvement phase.

In this work, we implemented the system level specification models in SpecC language, and the System-on-Chip Environment (SCE) [26] is used for component allocation, architecture mapping, and refinements. The integration process and the modified power-aware SCE tool is described in Chapter 4.

Although we picked SpecC language and SCE as the design environment, the concepts of MAVO can be used in SystemC or other design frameworks similarly. Moreover, each module of MAVO, *Monitor*, *PowerAnalyzer* API, *Annotator*, and power-performance *Optimizer*, can be used separately as needed or integrated into other tools as well.

The design flow in MAVO starts with an architectural model. In this work the architectural model is generated using SCE tool. When the architecture model is ready, *Monitor* produces power and timing traces of the architecture model. Next *Annotator* converts the model to a time and power aware model, in which every basic block of the model is annotated with power and performance information defined by utilizing *PowerAnalyzer* API. The power and time aware model is simulated in order to generate the power and performance estimation reports in addition to simulation result of the model. Followed by reports, the power and time *Optimizer* applies power management mechanisms and allows the designer to investigate the trace both numerically and visually throughout the simulation.

MAVO can generate the power and timing traces of all units and behavior of the system. All the trace files can be visualized with different sampling frequency and any combination base on designer desire.

In summary, once the architecture model is ready, the main power estimation and optimization steps are profiling the model, annotating power and time related functions to the design, the power and performance observation and analysis, and finally optimization.

2.1.1 Profiling

The design process of embedded systems starts with the original specification model of the design. This model contains the system functionalities only, without any timing or architectural information. In order to evaluate power dissipation, the specification model is profiled by means of capturing its energy consumption activities. Similar to SpecC Profiler [16], our power monitor is a design-oriented profiler that extracts the energy activity details on operations, communications, and memory characteristics of each behavior.

Monitor captures the execution number of each basic block by simulating the specification model. The operations and statements of each block is collected by traversing the internal representation of SpecC models. Then using the execution counts and operations set of each basic block the proposed retargetable profiler automatically annotates the design source code to generate both *static* and *dynamic* reports. These reports assist designers to comprehend the computation cost of each behavior in the design, moreover reports are annotated to the specification model for further power and performance analysis during power analysis and design exploration phase.

In *static* report, the evaluation is based on pure code, while in a *dynamic* report the actual number of executed operations and statements during simulation is taken in to account. For power estimation, the *static* report of each basic block within the model is used which contains sufficient information on the ESL model running its application. Monitoring these basic blocks is also shown to be a fine level of granularity for performance estimation in [16]. Having a comprehensive profiler is a key factor in developing an accurate and fast system level power estimator. In this work we developed a profiler that summarizes all the execution counts of all expressions and statements with their associated types.

An example of a profiling report for a specification model is presented in Figure 2.2. In this model, two instances of behavior *A*, *A1* and *A2*, are running in parallel with an instance of behavior *B* named *B1*. Behavior *Main* contains *A1, A2* and *B1* as child behaviors; hence,

```
behavior A()
{
 void main()
 {
  int x;
 for(x=1;x<11;)
 {
    waitfor 1;
    x++;
  }
 }
};
behavior B()
{
 void main()
 {
  float y=100;
  do{
     waitfor 2;
     y/=2;
  } while(y>10);
  y++;
 }
};
behavior Main()
{
 A A1,A2;
 B B1;
 int main()
{
 par
 {
  A1.main();
  A2.main();
  B1.main();
 }
   return 0;
  }
};
```

behavior: A

executed : 2	**dynamic**	**static**
constant, *int*	44	3
identifier access, *int*	44	3
post-increment, *int*	20	1
less than, *int*	22	1
assignment, int	2	1
expression statement	20	1
for statement	2	1
wait for statement	20	1

behavior: B

executed: 1	**dynamic**	**static**
constant, *int*	12	3
identifier access, *float*	9	3
post-increment, *float*	1	1
greater than, *int*	4	1
division-assign, *float*	4	1
expression statement	5	2
do-while statement	1	1
wait for statement	4	1

behavior: Main

executed: 1	**dynamic**	**static**
constant, *int*	57	10
identifier access, *int*	44	6
identifier access, *float*	9	3
post-increment, *int*	20	2
post-increment, *float*	1	1
less than, *int*	22	2
greater than, *int*	4	1
assignment, *int*	2	2
division-assign, *float*	4	1
expression statement	25	4
do-while statement	1	1
for statement	2	2
return statement	1	1
par statement	1	1
wait for statement	24	3

Figure 2.2: Dynamic and static profiling report of behaviors *Main, A* and *B*

its profiling report covers its own operations and expressions plus its child instances.

Monitor generates both static report, by traversing the pure code, as well as dynamic report through simulation. For instance in behavior A one *post-increment* operation is captured in the code and it is reported in the static reports of A and *Main*, however after simulation 20 *post-increment* operations were executed in total in the *for* loop and in two instances of behavior A.

As it is shown in Figure 2.2, each operation is profiled with regards to its data type. For instance in behavior *Main* the *post-increment* operation is profiled for both *integer* and *float* data types separately. The Monitor report also shows the number of times that behaviors A and B are executed, along with their operations and corresponding data type.

2.1.2 Power Analyzer

To evaluate the power consumption of the design at the system level, we insert the power activity information in to the system-level model. The power activity information is calculated by applying the obtained profiling information to the power models. In order to evaluate power dissipation, we designed a C++ API called *Power Analyzer* [73]. Power Analyzer functions are automatically attached to each basic block of the system model using the Annotator, to measure and monitor energy and delay. A graphical notion of basic block is shown in Figure 2.4.

Although power aware design is crucial in electronic system level design, SLDLs are not supporting this feature natively. The initial solution to this deficiency, which is still being used in some cases, was the use of spreadsheets. The spreadsheet approach is fast for average power evaluation, however verifying power dissipation over time and securing the system against power peaks is only possible when timing is taken into account [29]. There has been some research on extending SystemC or providing new libraries for power aware simulation. PowerSC [49] is a library proposed for power aware simulation of SystemC-based TLMs; it

can gather switching activity information during the simulation. TLM Power 3.0 [39] counts bit level activities in a TLM model. A similar approach is presented in PKtool [82].
A limitation of these approaches is that the annotation of power and timing functions to TLMs is performed manually, which is a tedious task for the designer and not scalable. Another drawback is that modifying the design with power aware functions changes the pure specification of the model.
In this work, we present Power Analyzer, a library for ANSI-C based SLDLs such as SystemC [55] or SpecC [34]. Power Analyzer allows power-aware design to be orthogonal to conventional performance-aware design. Power evaluations, measurements and visualization are developed and adapted for SLDLs. The proposed API utilizes all components switching activities, interactions and communication details generated by Monitor. The proposed extension and methodology for power modeling can easily be applied for any other SLDL as well.

To develop Power Analyzer, we defined *PowerMeter* as a virtual tool for power evaluation. *PowerMeter* is designed similar to an actual power meter that is used for power monitoring and measurements at the physical level. *PowerMeter* can monitor and measure power, and generate an on-line log of power activities during simulation. The collected information can be used for power optimization, power-aware schedulers, and ultimately increasing life-time and reliability of the system.
We have developed *PowerMeter* as an API and a library called Power Analyzer which offers power analysis along with timing log and graphical reports. *PowerMeters* are annotated to the specification model with energy dissipation and time consumption information. The implemented functions enable *PowerMeter* to monitor and measure power consumption over time, and within different system components or application segments. Table 2.1 shows an overview of existing time evaluation features along with added power features. In order to support power analysis the Power Analyzer library is developed with specific power and energy related units as well as functions for power consumption and monitoring. Owing to

Table 2.1: Time and power modeling

Features	**Time**	**Power**
Meters	sim_time *time*;	PowerMeter *pm*;
Units	SEC, MILLI_SEC,..	JOULE, MILLI_JOULE,.. WATT, MILLI_WATT,..

Consumption	wait(*event*);
	waitfor(*time*);
	do{...}timing{...}
		pm_consume_dynamic(*&pm,dynamic*);
		pm_consume_static(*&pm,static*);
		pm_consume_total(*&pm,dynamic,static*);

Monitor	*time* = now();
	time = delta();
		dynamic=pm_dynamic(*&pm*);
		static=pm_Static(*&pm*);
		total=pm_power(*&pm*);
		pm_display(*&pm*);
		pm_printPower(*&pm*);

Power Analyzer, the design exploration process can be initialized at the system level via verifying both timing and power constraints.

2.1.3 Power API

The Power Analyzer library uses *PowerMeters* (PM) to extract power dissipation information from the design, and perform power analysis. Each PM monitors static and dynamic power dissipation over time and with proper units. In order to support different operations over PM, such as power consumption, power monitoring and power constraints evaluation capabilities, multiple functions are developed in the Power Analyzer API which are presented in the following subsections.

2.1.4 PowerMeter

Each PM is capable of measuring power for its assigned part of the design. There is no limit on defining PMs nor associating them with particular part of the design. A PM can be defined for any block, component, and behavior of the code. The PM class objects can

be defined with type `PowerMeter` within the specification model code as:

```
PowerMeter pm;
```

2.1.5 Static & Dynamic Power

The PMs are treated as an actual power meter in this work. Therefore all the expected functionalities from a power meter are implemented as built-in features in the Power Analyzer API. For each PM the main required information encompasses the energy dissipated over time due to the switching activity and leakage. Hence these values are monitored and maintained internally for each PM.

```
long double Dynamic;
long double Static;
sim_time time;
```

2.1.6 Power Consumption

To represent the power dissipation in the system model a set of functions is presented; `pm_consume_dynamic` for spending dynamic power only, `pm_consume_static` for static power only, and `pm_consume_total` for spending both dynamic and static power.

```
void pm_consume_dynamic( PowerMeter *pm,
                         const long double Dynamic);

void pm_consume_static( PowerMeter *pm,
                        const long double Static);

void pm_consume_total( PowerMeter *pm,
                       const long double Dynamic,
                       const long double Static);
```

When these functions are called the `Dynamic` and/or `Static` energy values along with the associated time stamps are recorded for the specified PM. Then in the analysis phase this information is deployed to generate graphs and different forms of power reports.

2.1.7 Monitor Power Dissipation

To access the spent dynamic, static or total power value a set of functions are implemented. The function `now()` which returns current simulation time is already available in SLDLs.

```
void const long double pm_dynamic( PowerMeter *pm );
void const long double pm_static( PowerMeter *pm );
void const long double pm_total( PowerMeter *pm );
```

For monitoring dynamic and static power dissipation different functions are developed in Power Analyzer API. The provided functions are `print_pm_dynamic`, `print_pm_static` and `print_pm_total`, which print current dynamic, static and both respectively. These functions evaluate the power values and output the power report with proper units:

```
void print_pm_dynamic( PowerMeter *pm );
void print_pm_static( PowerMeter *pm );
void print_pm_total( PowerMeter *pm );
```

In order to evaluate the power dissipation over time a power-time diagram is a convenient solution for the designer. Function `pm_display` is developed to visually display power consumption during simulation:

```
void pm_display( PowerMeter *pm );
```

In Section 2.2 different examples of using function `pm_display` are represented in a case study on JPEG image encoder.

2.1.8 Power Analysis

When the model simulation is completed, PMs contain power behavior of the design. At this point, the designer can ask for different power information such as peak power, average power, display multiple PMs power dissipation in one graph, adding power dissipation of different PMs, or printing all static, dynamic dissipated values. Some auxiliary functions are implemented in order to allow these analyses. Function `pm_multidisplay` gets PMs as input and returns a merged graph of power dissipation over time of all the input PMs.

```
void pm_multidisplay(PowerMeter *pm1, ...);
```

An example of using `pm_multidisplay` is represented in Section 2.2.
During the power analysis, the user may want to add up power dissipation in multiple PMs and run further investigations. In `pm_add` a PM is returned that contains the summation of all input PMs

```
void pm_add( PowerMeter *result, PowerMeter *pm, ...);
```

For user convenience, in order to get a general report of all PMs the `pm_report_power` function can be used. It iterates through all PMs and prints the static, dynamic, total, average and total time spent in each PMs.

```
void pm_report_power();
```

Apart from the above mentioned functions, Power Analyzer API provides functions so PMs can be initialized, reset, or set to certain values for dynamic power, static power and time. The `pm_assign_dynamic`, `pm_assign_static`, `pm_assign_total` functions are used to initialize or assign values to a PM at time `t`.

```
void pm_assign_dynamic( PowerMeter *pm,
                        sim_time t,
                        const long double Dynamic);
```

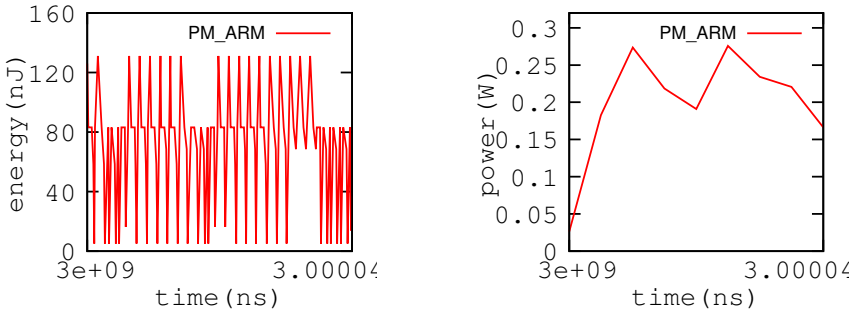
```
void pm_assign_static( PowerMeter *pm,
                       sim_time t,
                       const long double Static);

void pm_assign_total( PowerMeter *pm,
                      sim_time t,
                      const long double Dynamic,
                      const long double Static);
```

To reset the PM power values `pm_reset` can be called.

```
void pm_reset(PowerMeter *power);
```

2.1.9 Energy & Power

Each PM tracks the energy consumption due to switching activity and leakage. So the user can monitors how energy is spent at each PM over time and has the option to access the trace files with any requested timing resolution. An example of energy consumption at PowerMeter `pm` is presented in Figure 2.3(a). The power behavior is a direct function of



(a) Energy consumption (b) Power dissipation

Figure 2.3: Energy consumption and derived power dissipation

energy consumption over time and can be obtained easily from energy consumption infor-

mation. The Power Analyzer API assists the user to generate any format of power report and diagrams for needed simulation intervals and timing resolution. An example of power dissipation at PowerMeter `pm` is shown in Figure 2.3(b), where the average power dissipation is obtained from energy consumption with a sampling frequency of 1 milli second.

2.1.10 Annotator

In order to perform power and timing aware simulation, and design exploration the collected traces are annotated to the structural model. In order to maintain accuracy, the traces are associated with every basic block of the model. Therefore, an *Annotator* is designed to insert power and timing information. Nevertheless, the Annotator can support the annotation with higher granularity of behaviors and components as well. The back annotated information includes execution delay, static, and dynamic power dissipated within the corresponding basic block, considering the type of the design component that block is mapped to, its configured operational mode, as well as the communication transactions through the assigned communication unit. In order to process these values, Power Analyzer API [73] is applied. The Annotator is linked to Power Analyzer API in order to apply the power functions and use generated values for annotations. An example of a basic block annotated with time and power information is presented below.

```
{//basic block: Bi
 {
  Label i:
   waitfor 132 NANO_SEC;
   pm_consume_total(PowerMeterA, 3.2 MILLI_JOULE,2.5 MILLI_JOULE);
 }
 ...
 d++;
 Ch1.send(d);
```

}

As shown, the *pm_consume_total* function represents the amount of power spent in basic block B_i, in form of dynamic and static power, as well as the *PowerMeterA* that monitors B_i power dissipation. MAVO can support the dynamic and static power monitoring separately, so the designer can focus on any of them as needed. The static power represents leakage power, and because of shrinking in transistor size to sub-micron, static power needs to be investigated as carefully as dynamic power.

To evaluate the power consumption of the design at the system level, we insert the power activity information into the system-level model. The power activity information is calculated by applying the obtained profiling information to the power models. Power functions are back-annotated to every basic block of the system model to measure and monitor energy and delay. A graphical notion of basic block is shown in Figures 2.4. *PowerMeters* are also automatically attached to behaviors, architecture components and globally based on user preferences as follows:

- **Power Meter per Behavior**

  In this case, a power meter is attached to each behavior of the design. During the design, the system designer may want to analyze each behavior in terms of computation, power and performance, compared to the rest of the system. This information may also be used to evaluate the peak power and power hungry behaviors of the design for possible thermal issues. This information allows the designer to balance the computation of the system and adjust the design at the system level, where altering and re-evaluations are quick and easy.

- **Power Meter per Architecture Component**

  Here power meters are assigned to each processing element of the design. In order to explore the design options, different architectures may need to be evaluated. Power

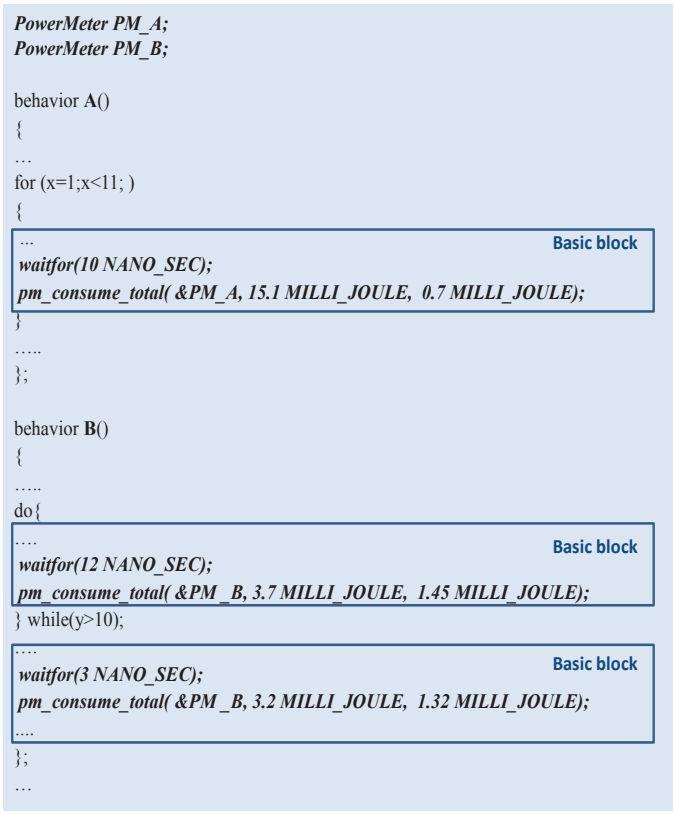


Figure 2.4: Power annotated system model with powerMeter for every behavior

dissipation of each component is monitored using the profiling information of the behaviors mapped to the component and shown by the corresponding meter.

- **Global Power Meter** In this option, a single power meter is attached to the entire ESL model, which measures the total power consumption of the design; this can be used to quickly determine if the design meets the overall power constraints of the target system.

Power meters are designed to capture dynamic and static power dissipation of their assigned component based on the power model. These components can be any CPU or hardware accelerator. In addition to the power meters, the *waitfor* [16] and *pm_consume_total* functions for time and energy consumption are automatically annotated to each basic block of the

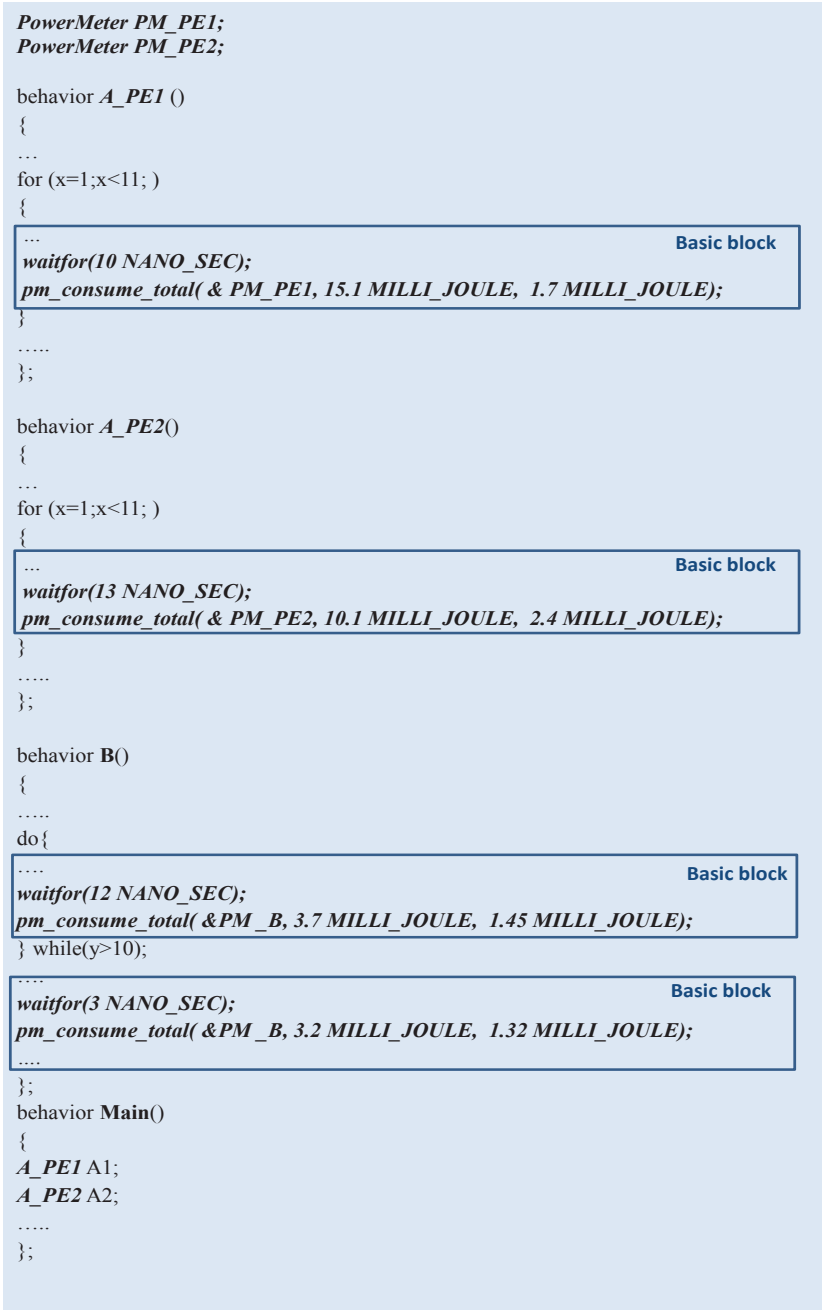


Figure 2.5: Power annotated system model with power meter per PE

design. The *pm_consume_total* functions virtually spend energy and represent the energy consumption during execution.

Specifically, the *pm_consume_total* function for a basic block b is:

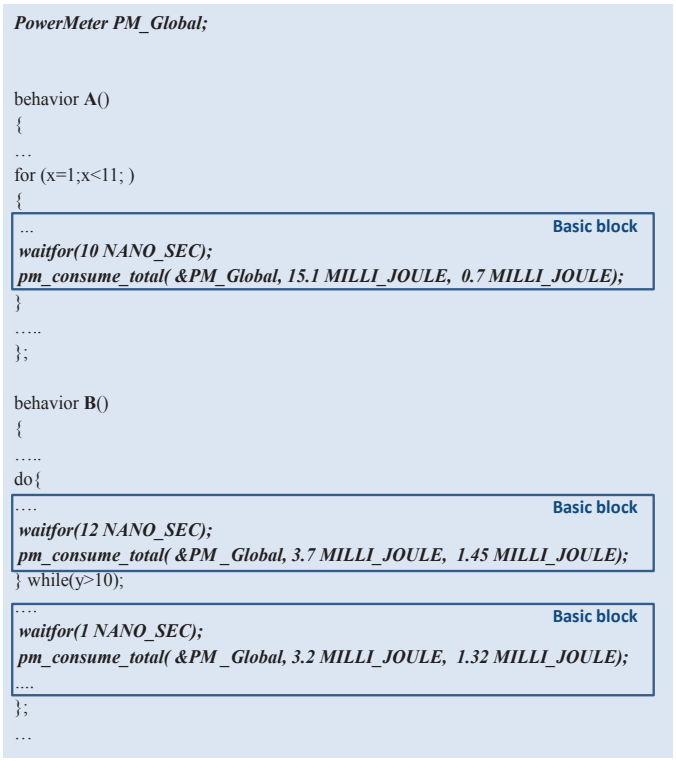


Figure 2.6: Power annotated system model with global powerMeter

$$pm\_consume\_total(PowerMeter_i,\ DynEnergy_b,\ StaticEnergy_b)$$

where $PowerMeter_i$ can be one of the three types of power meters. $DynEnergy_b$ and $StaticEnergy_b$ are the dissipated dynamic and static energy, respectively. In order to employ the suggested default power models, the dynamic energy of each basic block is computed directly by applying the number of operations reported by the profiler to the power models:

$$DynEnergy_b = \sum_{j,k} OpCount_{jk} \times OpEnergy_{jk} \tag{2.1}$$

where the $OpCount_{jk}$ is the number of operations j with type k and $OpEnergy_{jk}$ is the energy consumption of that operation derived from the power model. For static values, the execution time of the block ($time_b$) is divided by the total static energy ($Energy_b$) spent at

each behavior:

$$StaticPower_b = Energy_b \ / \ time_b \tag{2.2}$$

In Figures 2.6, 2.4 and 2.5, the power annotated specification model is presented for the profiled code shown in Section 2.1.1. The specification model is presented for the three different formats of power meters.

At this step the profiling information and power models are applied to Equation 2.1 and Equation 2.2, and the resulting energy dissipation is automatically annotated to the model via the *consume* functions. Similarly, the performance results from [16] are annotated automatically to the design using *waitfor* functions. For inserting the power meters per PE, the annotation of behavior A is implemented differently from the other two types of power meters. Since behavior A is mapped to two different *PEs, PE1* and *PE2*, the *consume* functions use different power models, *PE1* and *PE2*. To resolve this problem, new behaviors with the same functionality as behavior A are inserted and named corresponding to their allocated *PEs, A_PE1* and *A_PE2*. All instantiations of behavior A are modified accordingly, as shown in Figure 2.5. The duplications are performed automatically by our *PowerMeter* API during power meter insertion without any interaction by the user. The dissipated power and energy can be monitored both numerically and graphically using the available power functions of the Power Analyzer API.

2.2 Case Study: JPEG Image Encoder

The proposed MAVO framework is implemented as described in the last section. Here MAVO is utilized for monitoring power consumption in JPEG, as a real-life application. The MAVO is applied at global, PE, and behavior levels.

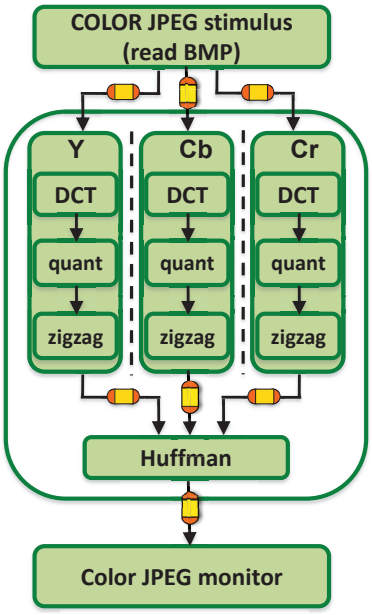


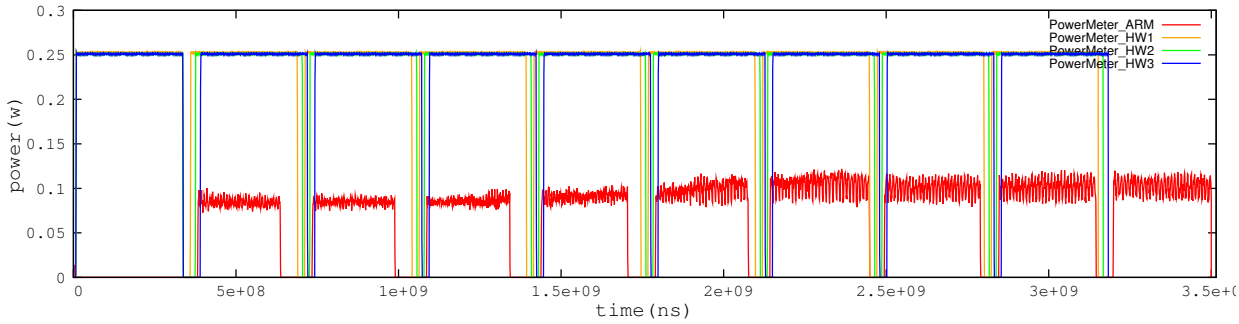
Figure 2.7: JPEG image Encoder [17]

Our case study uses the JPEG image encoder model shown in Figure 2.7. The stimulus reads a BMP color image with 3216x2136 pixels and performs color-space conversion from RGB to YCbCr. Since encoding of the three color components (Y, Cb, Cr) is independent, our JPEG encoder performs the *DCT*, *quantization* and *zigzag* operations for the colors in parallel, followed by a sequential *Huffman* encoder at the end. The image is divided in 9 strips and fed in to JPEG model. The JPEG monitor collects the encoded data and stores it in the output file.

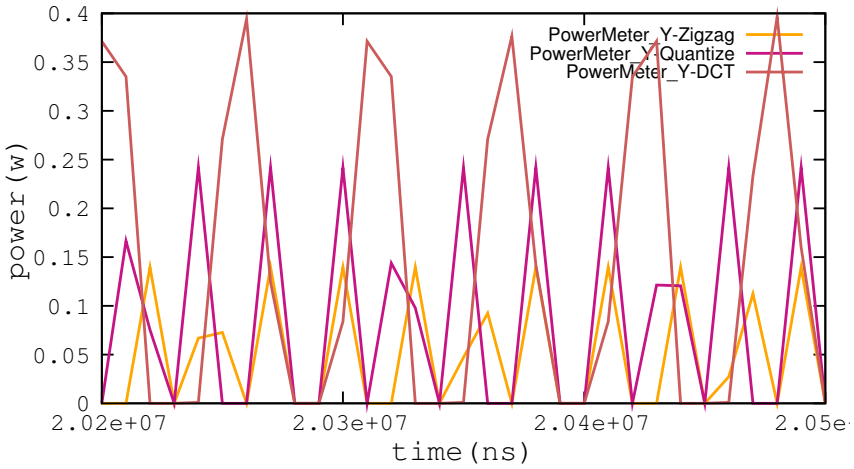
The JPEG model is examined on an ARM-based processor with 3 custom HW units. The 3 color components; Y, Cb, and Cr are mapped to separate HW units, along with their sub-behaviors (DCT, Quantize, Zigzag). All units are communicating through the AMBA BUS.

To comprehensively study power dissipation, we started with the specification model, and applied architecture, scheduling and communication refinement to the model, with increasing amount of implementation detail.

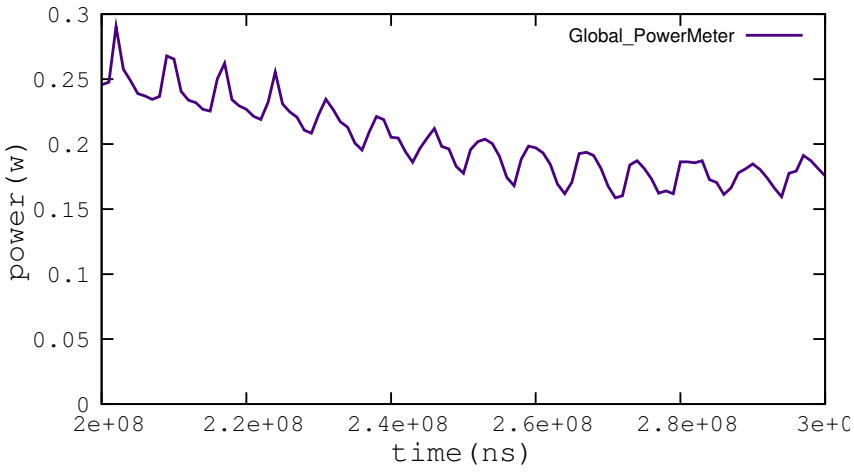
In order to control the size of power and energy log files, and adjust the precision of the



(a) Power dissipation at each PE



(b) Power dissipation at behavior level for behaviors mapped to HW1 (Y-DCT, Y-Quantize, Y-Zigzag)



(c) Average global power dissipation

Figure 2.8: Power dissipation of JPEG Image Encoder visualized and optimized by *PowerMeters*

analysis, the user can pick the sampling frequency. The user can also specify any simulation intervals to monitor as well. Moreover, MAVO supports merging the graphical reports or stacking up the power dissipation values in different PowerMeters over time.

Figure 2.8(a) shows the power dissipation in each design elements over the whole simulation time with sampling frequency of 1 millisecond. As it was expected from the JPEG model defined in Figure 2.7, where the Y, Cb and Cr are running in parallel, the custom HW units are executing them in parallel as well. The Huffman encoder, which is mapped to the ARM processor, begins working once the Y, Cb and Cr color processing is over for a strip within all HW units. The encoding process is divided in to 9 different steps, through 9 sub-images, and the power dissipation intervals of design elements in Figure 2.8(a) reflects the same behavior. In Figure 2.8(b) power dissipation in the Y-DCT, Y-Quantize, and Y-Zigzag, which are the leaf behaviors of Y is shown. The sampling frequency is 10 microsecond. All these behaviors are mapped to HW1. As it shown, the user can easily pick any behavior as well as any interval for power analysis. Using the power reports of each behavior, the user can easily verify the active behvaiors at each PE, computationally expensive behaviors, the working intervals of each behavior, and modify the design model rapidly if needed. The MAVO allows merging the power reports in to one graph based on the user selections.

The global PowerMeter monitors the average power dissipation of the whole system. Figure 2.8(c) represent the average power for selected simulation interval in JPEG application. The PEs, global and behaviors power dissipation reports support monitoring and analyzing the system for power and performance, and provide a platform for initiating power optimization. The power estimation API can also provide the energy dissipation graph for each power meter over time. For example, Figure 2.3(a) displays the energy dissipation graph of a global power meter in JPEG with 9 sub-image. As shown in the diagram, for each image, the energy dissipation increases during the complex encoding process for each sub-image.

Chapter 3

Accuracy and Fidelity Evaluation

In this chapter MAVO framework is verified for efficiency and correctness. The used metrics for evaluating MAVO framework are speed, absolute accuracy and fidelity. First we are looking in to these three metrics and their concepts in system-level design, then we used JPEG encoder, MP3 encoder and H.264 codec as real-life applications to evaluate the MAVO against these metrics.

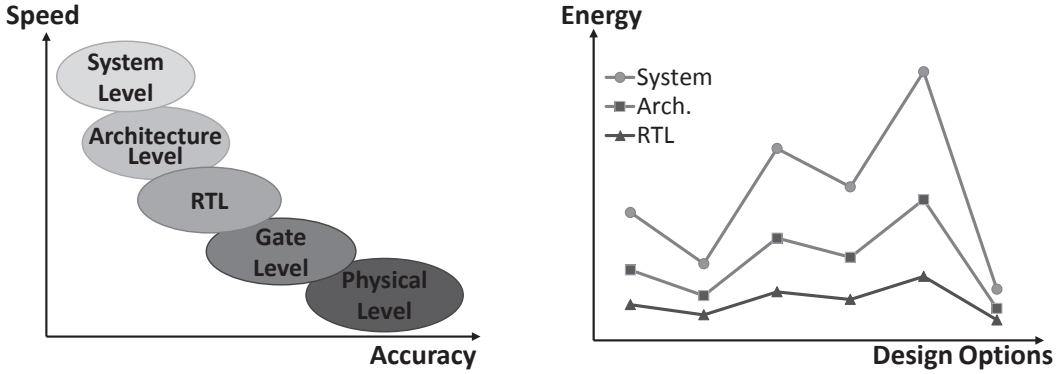
3.1 Evaluation Metrics

For the past few decades, semiconductor capabilities have been improving as Moore's law predicted. Transistor size has been shrinking and technology size will be less than 20nm in the near future. These improvements enable the designer to come up with more complex systems. However, this has made power dissipation a major design obstacle. , particularly in mobile and battery powered devices. The fact that power dissipation in small technology sizes increases due to high leakage power makes power optimization a primary target of the design process. On the other hand, the trend for battery-powered mobile communication

and all-purpose devices in the market force the semiconductor designers to consider power optimized solutions and ultra low-power design.

Conventionally, power consumption is considered in the later stages of the design process, like the architecture level [54], RTL [1] [67], gate level , and physical level, where detailed information about the design is available. The lower levels power estimator's absolute accuracy is highly accurate but the speed is significantly low since the simulation and evaluation time are high and often beyond the time-to-market requirements.

To tackle the long simulation time as well as avoiding time consuming design modifications at lower levels, designers are changing the level of abstraction to the system level, typically by means of trading accuracy in favor of speed. Figure 3.1 shows a prospect of power estimation at the system level. The speed-accuracy trade-off in power estimation at different design levels is demonstrated in Figure 3.1(a). The accuracy and time trade-off is the main challenge in power estimation. Here, a powerful and automated API for system level early estimation, as proposed in this work, can shorten the design cycle of low-power systems tremendously. The earlier the optimization starts, the more efficiently devices can be produced. Consequently, design constraints such as power, performance and die size are ought to be taken into account from the early stages of the design flow.



(a) Speed-accuracy trade-off in power estimation

(b) High fidelity energy estimation at different levels of design

Figure 3.1: Power estimation at different design abstraction levels

The System level is the starting point for design constraints characterization as well as

design space exploration. Therefore design decisions, such as component selections, Hardware(HW)/Software(SW) partitioning, communication schemes, number of cores, and power reduction techniques, are ideally all made at the system level. Here, it is critical to make correct decisions. To achieve this goal, a structured ESL tool suite is required to perform assessments. In order to select the best design options at the system level, *relative* accuracy and high *fidelity* [33] are essential. Figure 3.1(b) presents the notion of power estimation from a desirable system level estimator where the comparative relations of the design options are accurate. Thus, our goal in this work is to develop a system level performance estimator with a high level of *fidelity*.

A practical platform for design space exploration at system-level should satisfy power and performance as primary design constrains, while keeping designers away from RTL modeling and lower level design implementation details. The best solution for designer is a system-level EDA tool, capable of optimizing power and minimizing performance while spending minimum effort.

For system level power estimation, the two available design inputs are 1) the specification model implemented in a System Level Description Language (SLDL), such as SystemC [55] or SpecC [34]; and 2) the power models of different system components, such as processors and IPs. Many studies have been performed on characterizing power dissipation for memories [27], communication channels [47], and processing elements (PE) [54] using experimental and statistical analyses at lower levels, by actual measurement or by applying power model builders, such as PowerMixer [77]. The fidelity and accuracy of the system level power estimation directly depends on available functional information and extracted power activity details in the design, as well as applying effective power models.

3.2 Power Modeling

To work with the proposed automated power estimator, any power model can be applied during IP integration and mapping to the behaviors. Therefore, power characterization and modeling is not the main focus of this work. However we used a simple power model generation method to generate some default power models. *relative* accuracy, rather than absolute accuracy.

Our system level power models are based on power reports and simulation information from

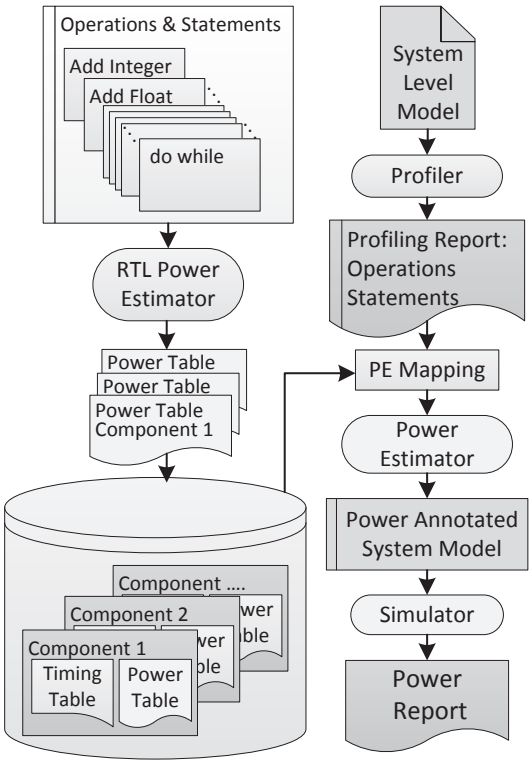


Figure 3.2: Power Estimation Flow

lower levels of design. In general, the accuracy of power evaluation is compromised by high level estimation approaches. However, as it is mentioned earlier the main concern at system level is to make the correct design decisions rapid and early in the design cycle. Therefore, it

is adequate that system level evaluations schemes, provide precise *relative* accuracy, rather than absolute accuracy.

Here, we present a default power model that is a combination of instruction and function based power modeling approaches. The main idea behind these power models, is similar to power modeling presented in Tiwari et al. [78]. Later in the experimental results we show that our proposed power API is able to deliver high fidelity even with these basic power models. In these models, each expression and statement has been measured using power simulators [62] [19]. This process is only performed once for each PE and the resulting power tables are added to the database as shown in Figure 3.2. During design space exploration within SCE, each PE power model is automatically added to the design while mapping design to PEs. The SCE allows the design allocation and mapping with its *sce-allocate* and *sce-map* features. In order to support the MAVO framework these two features are modified to cover the power models and power related configurations of the design components as well.

In our studies, we have used ARM-based and Intel Nehalem processor architectures. Each expression and statement in the source code of the model owes a dynamic and static energy consumption value. The dynamic energy is the energy spent as *dynamic switching* and *short circuit* power, and static energy represents the energy dissipation due to *leakage*.

Table 3.1: Dynamic and static energy values (nJ) for operations & statements for ARM7 processor

Operations Expressions	**Types**					
	integer		**float**		**long long integer**	
	Dynamic	*static*	*Dynamic*	*static*	*Dynamic*	*static*
Add	4.9	0.07	6.0	0.1	13.8	0.2
Division	57.0	1.3	82.1	2.3	298	6.9
do while	6.7	70.1	6.7	0.1	6.7	0.3

Table 3.1 shows a section of the generated energy tables for an ARM7 processor. In order to generate dynamic and static values, each expression/statement is simulated multiple times. Furthermore, simulations are statistically analyzed through regression analysis to even out the effect of cache misses, pipeline stalls, or any other situation that possibly increases the

simulation time as well as energy consumption. For instance, the integer *"add"* operation has been tested with 10^1, 10^2, 10^4, 10^5, 10^6 integer *"add"* operations and the energy consumption is evaluated for each unit of the target architecture. Figure 3.3 shows the energy consumption in different units for a series of *"add"* operations. The results show that for a large number of *"add"* operations the consumed energy will not vary significantly. Hence, we picked the average energy dissipation of 10^4 *"add"* operations as the reference energy.

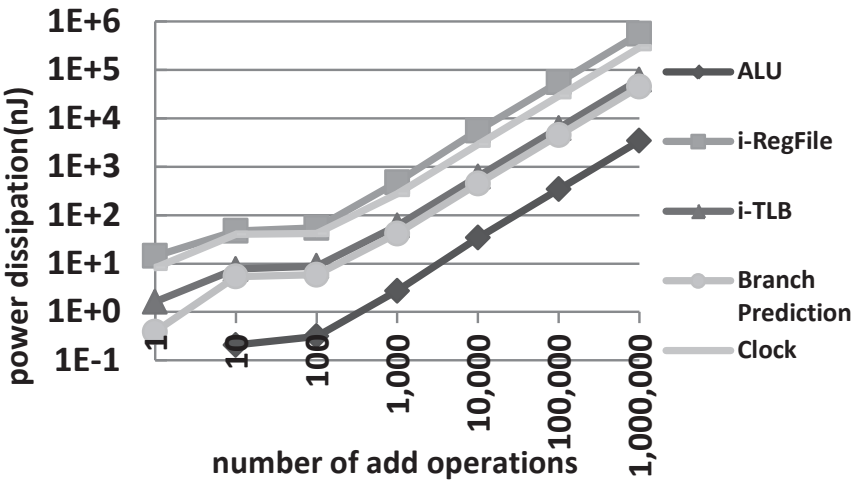


Figure 3.3: Power dissipation (nJ) for n *"add"* operations

Other expressions, statements, and operations were similarly studied for each type in order to obtain dynamic and static power parameters.

3.3 Experimental Result

In this section, we describe our experimental results and analyse the speedup, *fidelity*, and accuracy of *PowerMeter* with real life applications. We implemented the profiler and *PowerMeter* API, as well as a platform for automated back annotation of power meters, *consume* functions, and *waitfor*. Thus, when the specification model is ready, the system designer can instantly evaluate the design by simply mapping the ESL system model to architecture

components and set the *PowerMeter* granularity.

For our experiments, we choose a JPEG image encoder application [17], a MP3 audio decoder [21] and a H.264 video encoder [41] and decoder [22] application, all specified in SpecC SLDL. To evaluate the *fidelity* and accuracy of the proposed power estimation method, we modified the ISS model of JPEG encoder to simulate on SimpleScalar [15] and measured the energy and power consumption using the cycle-accurate SimPanalyzer [62] for the ARM based architecture. The MP3 and H.264 examples are simulated using the SNIPER 5.2 [19] simulator and their power is evaluated using the embedded McPAT [54] against the Intel Nehalem architecture. We also applied these applications to our power estimator. All results are measured on a host PC with a 4-core CPU (Intel(R) Core(TM)2 Quad) at 3.0 GHz.

Our results are presented in Table 3.3 and Table 3.2. The comparison execution time shows significant speedup in *PowerMeter*, in comparison to the cycle accurate SimPanalyzer, as well as SNIPER, with the embedded McPAT power estimator as an architectural level simulator. The overall speedup shows that *PowerMeter* is about one order of magnitude faster than the alternatives.

Table 3.2: MAVO speedup experimental result for JEPG, MP3 audio decoder and H.264 video codec

Model		**Time(sec)**		**Speed Up**
		simPanalyzer	*MAVO*	
JPEG Enc.	Image1	52.7	4.04	13x
	Image2	634	4.12	154x
	Image3	287	4.08	70x
	Image4	134	4.07	33x
		McPAT	*MAVO*	
MP3 Dec.	Audio1	645.1	135.05	4.8x
	Audio2	682.6	137.68	5.0x
	Audio3	118.1	18.21	6.5x
	Audio4	612.1	128.35	4.8x
H.264 Codec	Video1	9967.54	479.85	20x
	Video2	3209.2	70.23	45x
	Video3	13942.63	191.29	70x
	Video4	+2hr	353.53	-

Furthermore, the *PowerMeter* is a system level power estimator and, with its low execution time, provides a practical platform for design space exploration. As shown in Table 3.2 for

Table 3.3: Power estimation experimental result for JEPG, MP3 audio decoder and H.264 video codec

Model		**Energy(J)**		**Error**	**Scaled Energy**		**Accuracy Error**	**Fidelity Score**
		simPanalyzer	*MAVO*		*simPanalyzer*	*MAVO*		
JPEG Enc.	Image1	7.3	8.76	+12.00%	0.0%	0.0%	+0.0%	100
	Image2	166.7	176.73	+6.20%	100.0%	100%	+0.0%	
	Image3	78.6	78.05	+1.60%	44.7%	41.3%	-3.5%	
	Image4	33.5	36.81	+9.5%	16.4%	16.7%	+0.3%	
		McPAT	*MAVO*		*McPAT*	*MAVO*		
MP3 Dec.	Audio1	1.81	2.72	+50.30%	92.4%	92.5%	+0.1%	100
	Audio2	1.94	2.91	+50.30%	100.0%	100.0%	+0%	
	Audio3	0.24	0.36	+50.20%	0.0%	0.0%	+0.0%	
	Audio4	1.74	2.64	+51.50%	88.2%	89.4%	+1.2%	
H.264 Codec	Video1	57.72	27.02	-53%	100.0%	100%	+0.0%	100
	Video2	7.16	6.56	-7.9%	0.0%	0.0%	+0.0%	
	Video3	20.21	15.56	-23.00%	100%	100%	+0.0%	
	Video4	-	33.74	-	-	2.17	-	

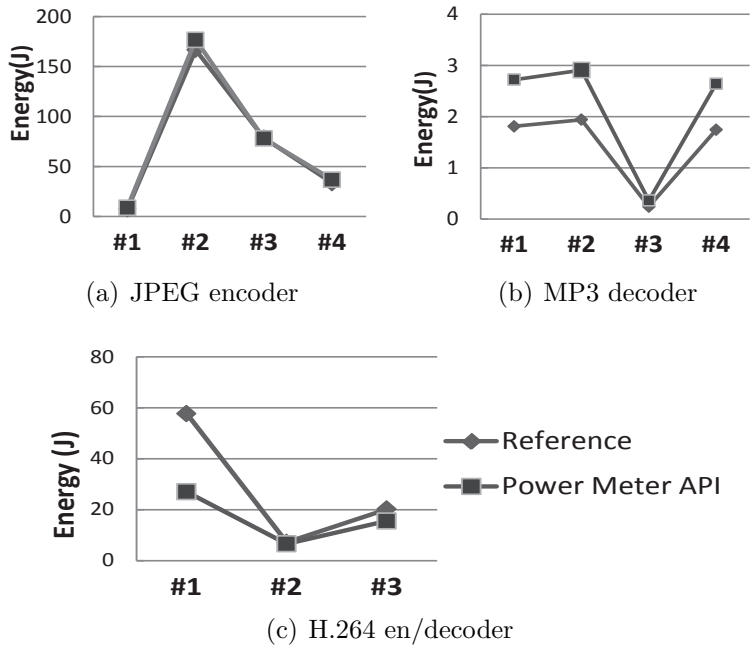
computationally expensive applications, a low-level simulator is not an applicable solution. For instance, power estimation of the H.264 application with 10 frames did not complete after two hours of runtime while its simulation ended in less than 6 minutes at the system level.

The error reports of simulated applications are as expected in Figure 3.1. The absolute accuracy is not as good as a low level estimator and goes up to 50% in MP3 application. This is simply owing to the fact that the accuracy of the estimated energy dissipation highly depends on the information captured during lower level simulations. Another reason is the accuracy of power models and the fact that the accuracy of power estimation directly depends on the accuracy of these models. In the applied power models, due to the fact that the models are not data-dependent and that even minor errors within the estimated power value for each operation will accumulate in total power estimation calculations, this results in higher errors specifically in large applications. For instance, in the MP3 example, the accuracy errors for all four audio streams are uniformly very close (only 1% difference). This clearly reflects the error in the original power model. However, the values of accuracy error using scaled energy for each sample application show the absolute maximum error is 3.5%, which reveals the high *fidelity* of our *PowerMeter* estimator.

The *fidelity score* for the system models are also calculated using the Equation 3.1 proposed in [51].

$$F_{score}(model_m)=100 \times \frac{2}{n(n-1)} \sum_{\substack{i,j=1 \\ i<j}}^{n} \mu_{ij} \tag{3.1}$$

Here, *fidelity score* F_{score} is calculated for system model m with n different design samples using predictor fidelity function μ. This function compares the referenced(i) and generated(j)



(a) JPEG encoder (b) MP3 decoder

(c) H.264 en/decoder

Figure 3.4: Power evaluation fidelity for system level and RTL

design samples under multiple criteria, and returns 0 or 1 accordingly. As shown in Table 3.3, this confirms the perfect *fidelity score* of 100 in all three of the applications.

The fast speed and high level of fidelity are the main achievements of our system level power estimator, which make SCE *PowerMeter* a solid starting point for power evaluations at the early system level.

Figure 3.4 shows the results of *PowerMeter* API against McPAT and SimPanalyzer power

estimators. As demonstrated, all pairs of curves, which represent energy values of different inputs for each application, all show the same shape, confirming the high fidelity of our approach.

Chapter 4

Power-Aware Design Space Exploration

Electronic system level design is presented as a powerful solution for embedded system design with all design complexities and the time-to-market constraints. The ESL design is also an effective starting point for power analysis and optimization and tackle high power dissipation. In this chapter we are presenting the SCE design space exploration tool extension for power-aware ESL design flow. First, we look in to SpecC language as the foundation of SCE, followed by brief description on SCE environment and its extended power features. Then we demonstrate power-aware SCE as applied to design of Canny image detector application. Results of the SCE-based design process approves the feasibility and efficiency of the power-aware SCE environment. Using the SCE exploration environment different models can be generated and verified against power and performance constraints within an hour.

4.1 SpecC Language

SpecC is a system level description language and methodology developed in Center For Embedded Systems at University of California Irvine[34]. This languages is designed for system level SoC design and presents all required features for this purposes. The SpecC is compiled with SpecC compiler called *scc*, and it is developed using SpecC Internal Representation (SIR) data structure [84]. This structure is composed of different classes to support SpecC's extensive facets for SoC design.

In contrast to SystemC SLDL which is C++ library and it is an extension to C++ language, SpecC concepts and features are defined for describing SoC models. The main extension of SpecC language in compare to ANSI-C languages are structural and behavioral hierarchy, separating computation and communication, synchronization, and timing.

In SpecC, every program is composed of different behaviors, channels and interfaces. These different classes are defined to support computations (behaviors), and communication (interface, channels) separately within the structure of the design model.

Conventionally in high level languages, a sequential execution order is applied during simulation, however in SpecC a behavioral hierarchy is introduced to allow behavior execution in sequential, parallel, FSM and pipeline format. The behavioral hierarchy enables the designers to implement application in compatible to hardware capabilities.

In order to support communication and sharing resources among concurrent behaviors multiple synchronization statements are provided. The *wait, notify* and *notifyone* statements are defined for this purpose.

Another required concept for describing SoC design models is timing. In SpecC two form of timing are introduced, the *waitfor* statements which are used for specifying the exact timing and *do-timing* and *range* which are mainly used for defining timing constraints.

Although SpecC languages was initially developed with high compatibility for SoC design,

the feature for integrating the notion of power in design models is missing in this language. In this work the Power API (Section 2.1.2) library is presented to support developing power-aware models.

4.2 System on Chip Environment

The System-on-Chip Environment (SCE) [26] is a SpecC based graphical tool for automated ESL design and suitable for rapid MPSoCs design space exploration. An overview of the flow is shown in Figure 4.1.

The graphical user interface of SCE is composed of a refinement user interface that allows the designer to perform PE allocations, behavior mapping, scheduling, network refinements along with a validation user interface to verify and validate the refined models.

The SCE design flow utilizes the *Specify-Explore-Refine* methodology with built in feature

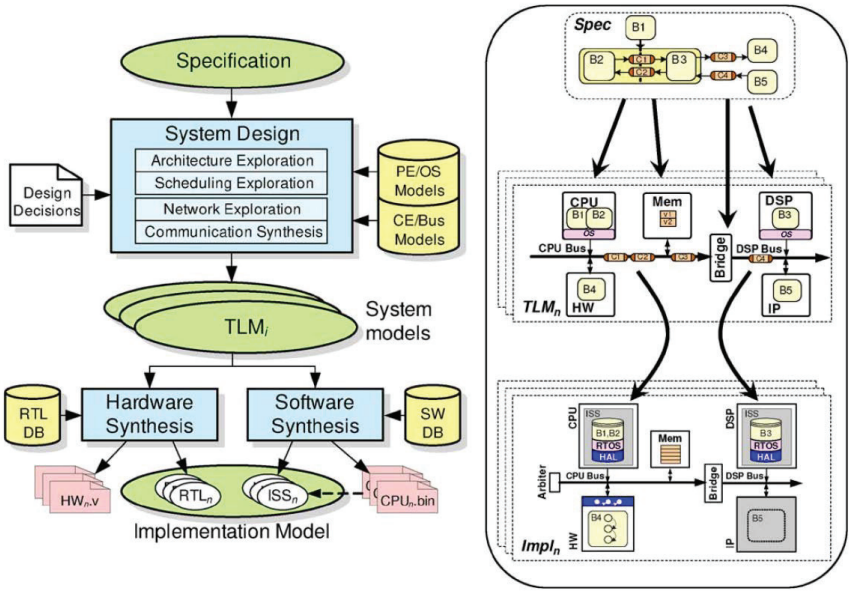


Figure 4.1: Top-Down System-Level Design Flow [36]

for exploring the design space. The SCE design flow begins with design specification model at system level, then the model goes through different refinement step down to hardware and

software implementation. At each refining step design can be evaluated and design space can be explored using a retargetable profiler and a performance estimator. The SCE automatically implement the specification model on a given target platform according to component allocation and mapping decisions. It also supports MPSoCs with different custom hardware, programmable software processors, IP blocks and memories, communicating through buses and communication elements such as transducers and bridges.

At the end of the design flow, SCE produces a transaction level model which both computation and communication part of the design can be synthesized to RTL models on the hardware side, the application tasks, middleware and bus drivers is automatically synthesized to binaries for the target processors [36].

4.3 MAVO Integration

The MAVO framework has been integrated into SCE to enable the power-aware design space exploration. In order to support power analysis multiple features has been added or extended in SCE:

- The retargetable profiler in SCE is extended to support all operations and statements with different data types
- The SCE Allocate and SCE Map features are adjusted to cover power related information of design components on top of the timing information.
- The SCE design components database is updated with power models. The power models are generated through the power modeling scheme presented in Section 3.2.
- The SCE is linked to *Power API* to support PowerMeters and power functions.

- The power *Annotator* module in MAVO is also integrated to SCE tool to support power-aware refining.

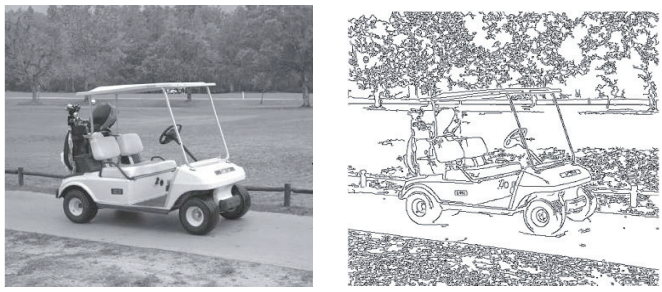
With added features and extentions the SCE is capable of power and performance aware design space exploration. In the following the Canny edge detector application is evaluated for power and performance.

4.4 Canny Edge Detector

4.4.1 Overview of Canny Edge Detector

The Canny application is designed for image edge detection which generates new image with all the edges of the input image. An example is shown in Figure 4.2. The original model was developed by Prof. John F. Canny in 1986 and our work is based on a reference implementation [18].

In this edge detection algorithm five primary functions are applied to input image. An



(a) Input image (b) Output Image

Figure 4.2: Canny Edge Detector [42]

overview of this edge detection structure is presented in Figure 4.3. These functions and their contribution are:

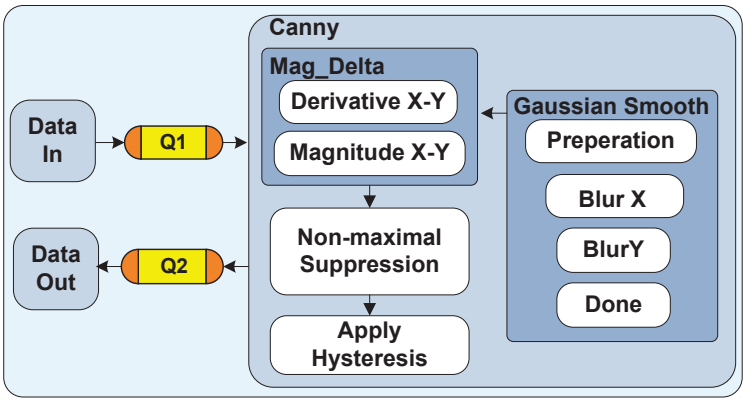


Figure 4.3: Canny Edge Detector Specification model [42]

- **gaussian_smooth** creates a gaussian kernel based on input parameter *SIGMA* (the standard deviation of the gaussian smoothing filter), and then used the kernel to filter or blur each pixel of the image to reduce the noise. The blurring occurs first horizontally and then vertically.

- **derivative_x_y** computes the first derivative of the image in both the x any y directions.

- **magnitude_x_y** computes the magnitude of the gradient - the square root of the sum of the squared derivative values.

- **non_max_supp** applies non-maximal suppression to the magnitude of the gradient image. The pixels which are not part of local maxima are set as non-edges.

- **apply_hysteresis** finds edges that are above some high threshold or are connected to a high pixel by a path of pixels greater than a low threshold. Parameter *TLOW* and *THIGH* specifies these two thresholds.

4.4.2 System Level Modeling of Canny Edge Detector

The first step of ESL design on canny is to create a specification model. We recoded the unstructured and sequential *C* reference code into SpecC model by works including encapsulating functions into behaviors, creating channels and hierarchy, and creating a testbench [42]. The resulting specification model is described as Figure 4.4 where *stimulus* sends incoming images to *platform*, I/O units(*din* and *dout*) send input to DUT *canny* and send output to *monitor*. DUT *canny* consists of five main behaviors performing the five functions of canny algorithm.

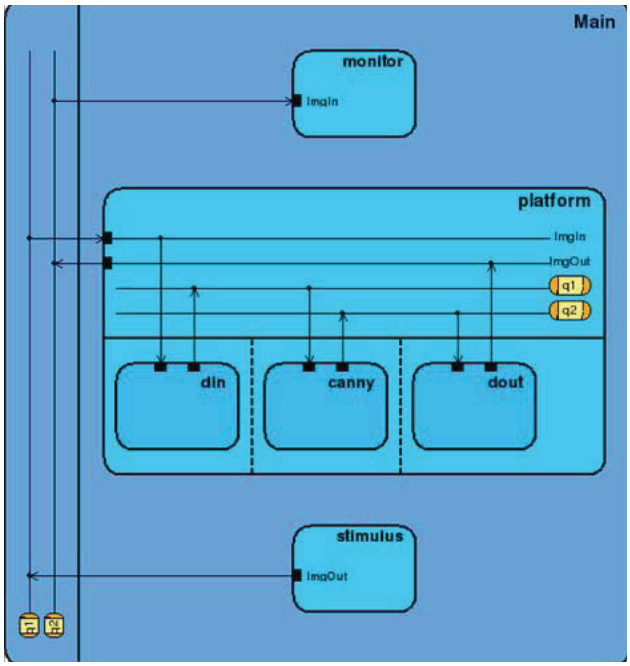


Figure 4.4: Stimulus, Platform and Monitor in initial Specification Model (Source [42])

4.4.3 Pipelined Canny

Canny image detector has limited concurrency within its algorithm and this does not allow to apply parallelism in behavioral hierarchy. We can, however increase the performance further by pipelining the edge detection algorithm in order to exploit and expose additional

parallelism across successive input images. This can be achieved by assigning a pipeline stage to each of Canny main functions. However in order to balance the workload of the pipeline stages to improve the pipeline performance, we need to consider the complexity of these behaviors.

In order to balance the pipeline stages we profiled the specification model using extended SCE profiler. The results are shown in Figure 4.5.

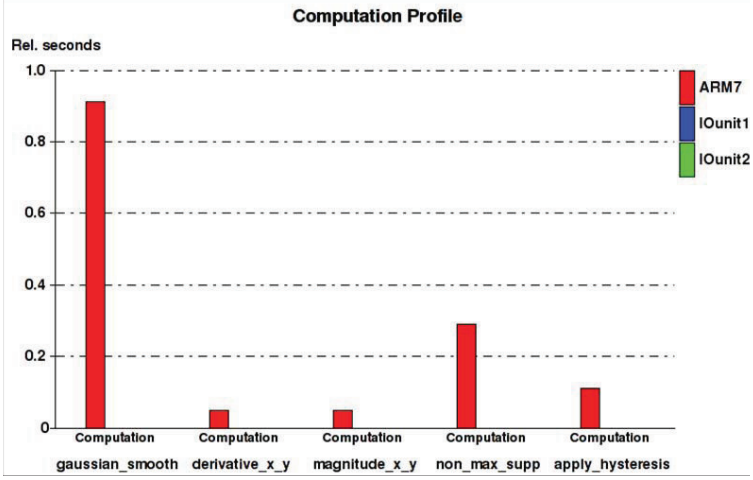


Figure 4.5: Canny Profile using SCE (Source [42])

As it is shown for a pure software solution of the Canny image detection running on an ARM7TDMI target processor, the *Gaussian* behavior takes more than 50% of total computation while the *derivative_x_y* and *magnitude_x_y* are together below 15%.

Hence, in order to yield better design, we modified the the design model structure. The *Gaussian* is splitted in to two separate behaviors, *Gaussian_Smooth_X* and *Gaussian_Smooth_Y* to reduce the computation load in *Gaussian* behavior. The *Gaussian_Smooth_X* and *Gaussian_Smooth_Y* are then mapped to two pipeline stages. Given these modification, the *Gaussian_Smooth_X* and *Gaussian_Smooth_Y* are still computationally more expensive in compare to other behaviors. In the next section we consider these critical behaviors for architectural decisions and possible performance improvements.

Additionally we merged the *derivative_x_y* behavior and *magnitude_x_y* behavior to a new behavior named *magnitude_x_y* and map them to single pipeline stage.

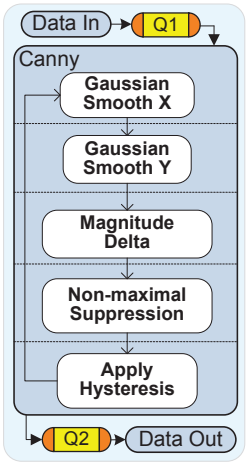


Figure 4.6: 5-Stage pipeline Canny Edge Detector

Figure 4.6 shows Canny with 5-stage pipeline structure.

4.5 Design Space Exploration

Given the specification model, we studied the Canny edge detector application on an ARM-based target platform. In the exploration process we evaluated the computational complexity of different behaviors and examined different architectural choices accordingly, to examine the power and performance benefits. The ultimate goal in evaluating multiple design option is to minimize the delay and power and find the optimal pareto.

4.5.1 Pure Software Implementation

We started the design and exploration process with the most basic target architecture. In this architecture an ARM based processor (*ARM7TDMI*) runs the main Canny edge detection top behavior.

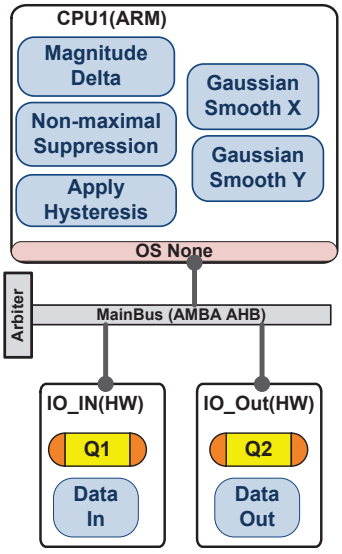


Figure 4.7: Pure software implementation

The *Stimulus* and *Monitor* are working as *DataIn* and *Dataout* behaviors. In addition to the actual edge detection algorithm running on the ARM, the processor is assisted by two hardware I/O units for image input and output. Therefore the *Data In* and *Stimulus* behaviors of the top level Canny design specification are mapped to *IO_IN* and *IO_Out* units, respectively.

Furthermore, note that the two FIFO queues for image input and output between Canny, Monitor and Stimulus have been each mapped into the corresponding hardware unit for implementation. Using SCE, the queues will be implemented as local and send and receive FIFOs inside each of the hardware I/O processors.

The ARM processor and the I/O blocks communicate over a single instance of an AMBA AHB local processor bus. The ARM processor is a master on its bus and the two I/O units are synthesized to connect as AHB slaves. As such, all communication between the ARM processor and the I/O units will be routed over the AHB bus. Specifically, Canny running on the ARM processor will read input image data from and write output image to the hardware FIFOs in the*IO_IN* and *IO_Out* blocks, respectively. All communication between the ARM

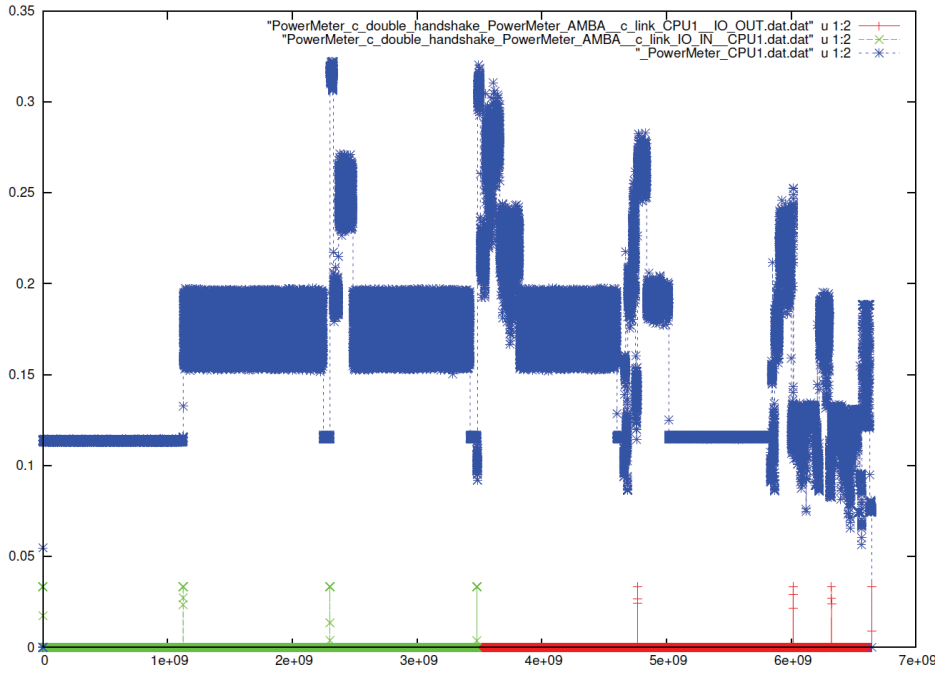


Figure 4.8: Power dissipation in pure software implementation (4 images)

processor and the I/O queues are implemented by mapping FIFO registers and link channels into the AHB address space using dedicated bus addresses and processor interrupt lines. This architecture is shown in Figure 4.7.

4.5.2 Hardware Acceleration

In Section 4.4.3 we started the exploration by evaluating the specification model behaviors and modify the design accordingly. In the optimization process the *Gaussian_Smooth_X* and *Gaussian_Smooth_Y* were chosen for hardware-assisted acceleration due to their high computational weight.

In order to evaluate multiple option for power and performance we examined the four different architecture with different mappings and hardware allocation. In the following we look in to these models in more detail with reports on performance and power along with their power

dissipation behavior over time.

Gaussian_Smooth_X Hardware Acceleration

The first selected architecture is to allocate a single co-processor for Gaussian_Smooth_X processor. This hardware processing element(*HW1*) works as one of the stage of the pipeline.

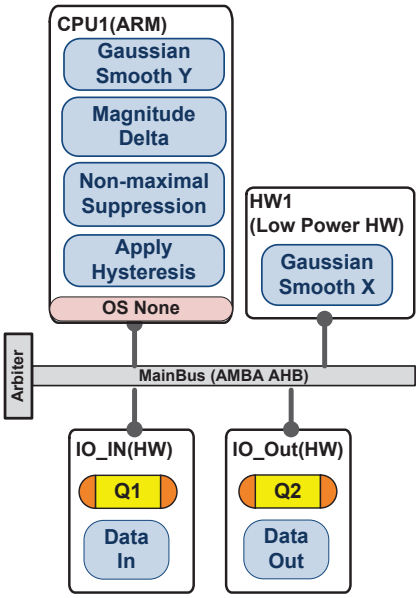


Figure 4.9: Synthesized Gaussian Smooth X hardware

The architecture of this design option(Canny HWX) is demonstrated in Figures 4.9.
As it is shown all other part of the architecture are similar to pure software implementation. Apart from the running the Gaussian_Smooth_X on separate PE, the communication between ARM software and *HW1* need to taken in to account for performance and power evaluation. The power dissipation in this architecture is presented in Figure 4.10. As it is shown from power dissipation graph Gaussian_Smooth_X process on *HW1* starts as soon as images are received from *IO_IN*.
From analyzing the slack times in *HW1*, it can be seen that Gaussian Smooth X process does not starts from image 3 and image 4 right after their previous images. This is simply due

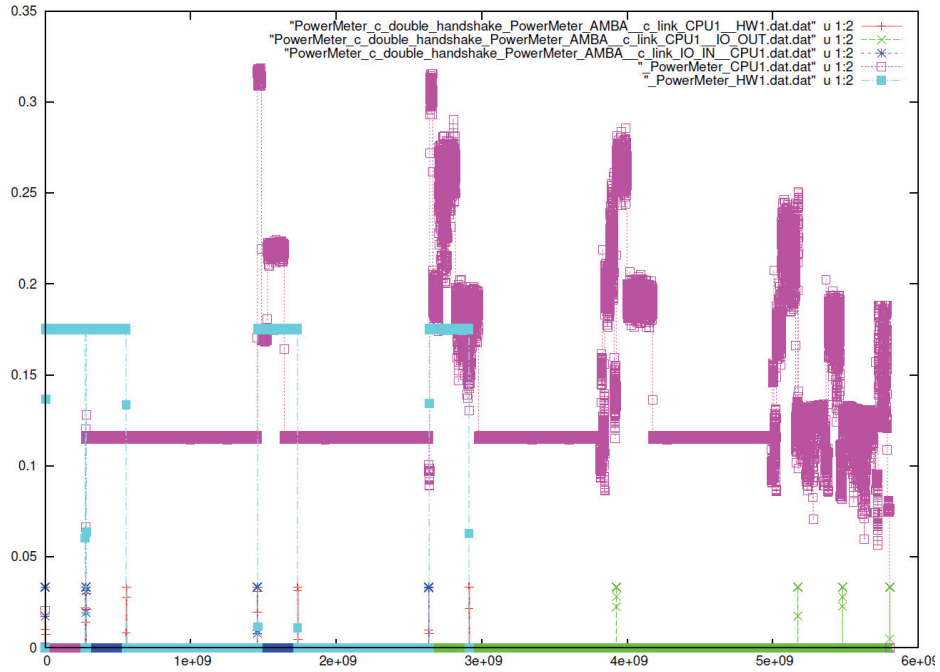


Figure 4.10: Power dissipation in Synthesized Gaussian Smooth X hardware architecture (4 Images)

to the fact that the other 4 stages of the pipeline are all mapped to ARM. Also the FIFO queues between the pipeline stages allows for only one image at a time.

Gaussian_Smooth_Y Hardware Acceleration

In this architecture the Gaussian_Smooth_Y behavior is selected for hardware acceleration. The resulting system computation and communication architecture is shown in Figure 4.11. The power dissipation behavior of the system over time is also presented in Figure 4.12. As a result when reaching the corresponding stage in the edge detection process, the ARM will send the input data to the HW1 component for *Gaussian_Smooth_Y* processing . The ARM software will then wait for result coming back from the HW1 before continuing the edge detection process.

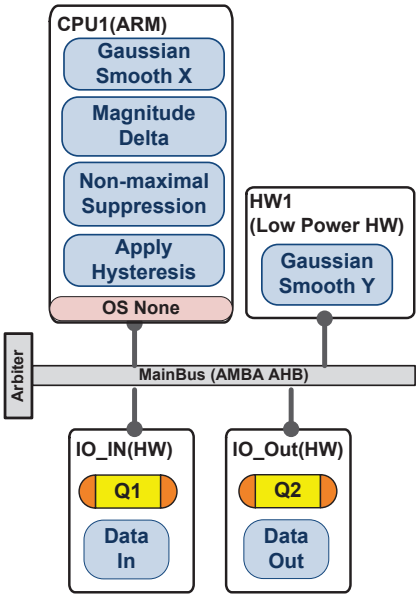


Figure 4.11: Synthesized Gaussian Smooth Y hardware

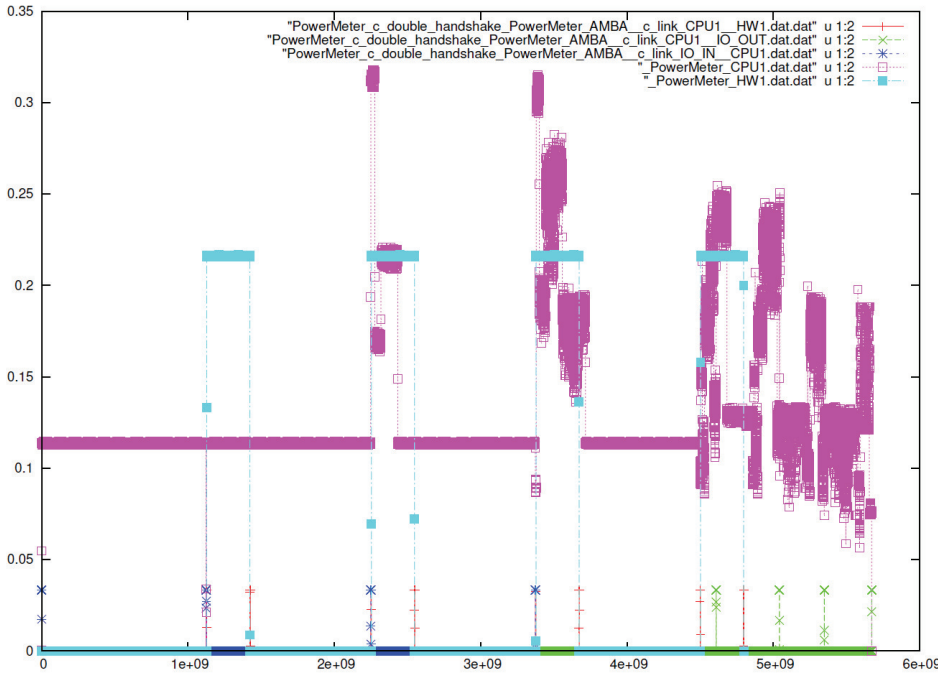


Figure 4.12: Power dissipation in Synthesized Gaussian Smooth Y hardware architecture (4 images)

The working periods of this stage clearly reflects the number of the images and communication time points.

Gaussian_Smooth_X and *Gaussian_Smooth_Y* Hardware Acceleration

The next reasonable step in hardware acceleration is to duplicate the custom hardware unit in order to provide dedicated co-processor instance for each of the Gaussian_Smooth behaviors, X and Y. In this architecture stage 1 and stage 2 of the pipeline are mapped

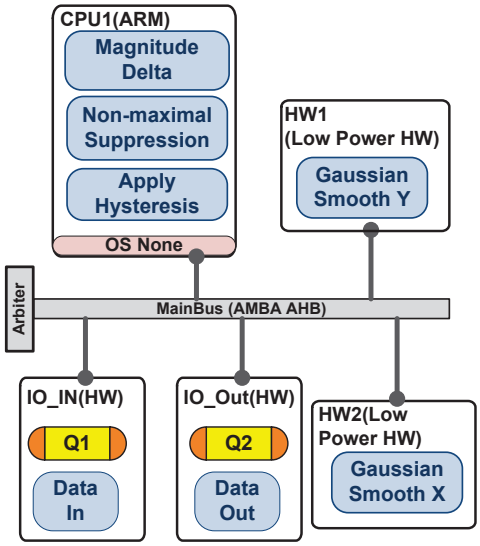


Figure 4.13: Concurrent Gaussian Smooth X and Y hardware

to two different custom hardwares. The resulting system computation and communication architecture is shown in Figure 4.13.

Figure 4.14 shows the power consumption of different components over the simulation time. The pipeline format of stage 1 and stage 2 of Canny edge detector is reflected in power dissipation in these two units. In this architecture the power dissipation in ARM processor has lower peaks and more distributed, since the computationally expensive stages are mapped to hardware accelerators.

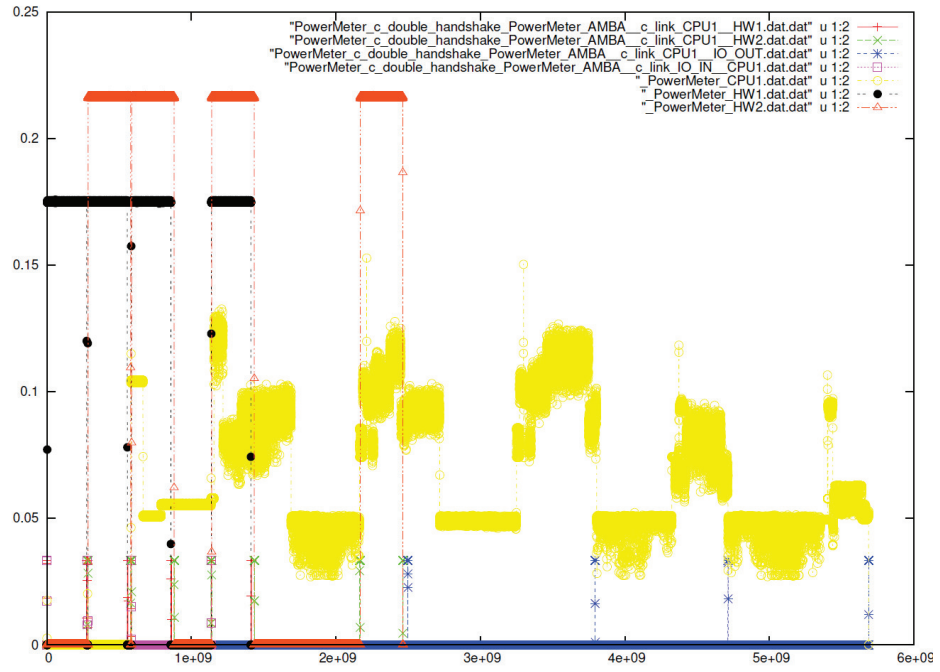


Figure 4.14: Power dissipation in synthesized Gaussian Smooth X and Y hardware architecture (4 images)

Parallelized *Gaussian_Smooth_X* and *Gaussian_Smooth_Y* Hardware Acceleration

In order improve the performance even further we used the Gaussian_Smooth_X and Gaussian_Smooth_X internal behavior. Theoretically these two behaviors can be parallelized on multiple cores. Here we are splitting these two behaviors in two function equally parallel modules. Each of these modules are now mapped to a custom hardware accelerator. Figure 4.15 shows the architecture with allocated PEs and behaviors mapping.

All the four accelerators are assumed to be connected as a salve to a common shared *IP Bus* instance that is connected to the main system bus via transducer. In the case of synthesizable custom hardware components, all four co-processors are directly connected as

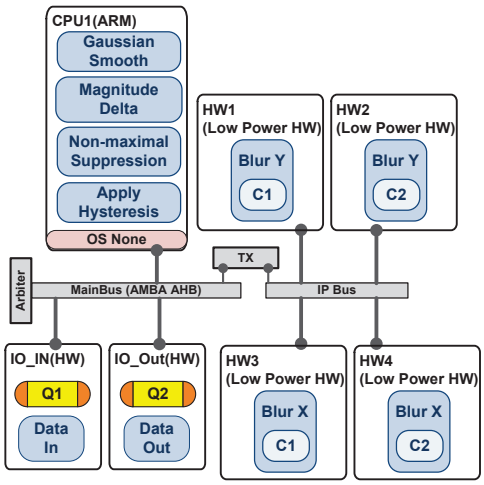


Figure 4.15: Concurrent Gaussian Smooth X and Y hardware

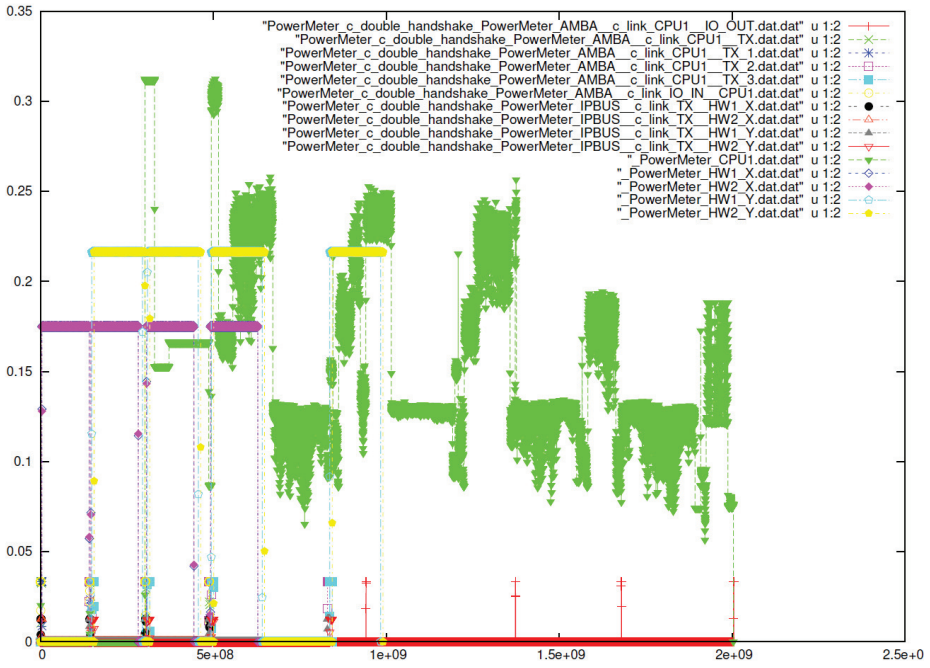


Figure 4.16: Power dissipation in Concurrent Gaussian Smooth X and Y hardware architecture (4 images)

and synthesized to become slaves on the AHB bus. In all cases, co-processors are slaves listening to and generating interrupts for a sole master ARM processor. As it is presented in power-time graph in Figure 4.16 the execution time has been reduced in cost of spending

more power. The other effective factor in power is communication.

4.5.3 Exploration Result

Going through the different exploration and refinement steps of the power-aware SCE design and tool flow, we realized the design implementation for all explored system architecture described in Section 4.5.1 and 4.5.2. Using the power-aware SCE's automatic model generation and refinement capabilities, the refined model annotated with power and performance information are generated. The text and graphical report of each design options was also captured and analyzed for further design space exploration. Moreover, power-time annotated models of each design option were generated within minutes and thanks to power-aware SCE tool it took less than an hour to investigate the design options.

The performance and power of the explored architecture options are presented in Table 4.1. The presented result are for the 5-stage pipeline canny under 4 images. As it it is shown

Table 4.1: Exploration results for Canny edge detector

Canny Architecture	Time (sec)	Power (mW)	Image (ms)			
			Image 1	Image 2	Image 3	Image 4
Pure Software	6.647	234.12	4,769	6,016	6,323	6,647
Canny HWX	5.804	232.355	3,934	5,187	5,493	5,818
Canny HWY	5.672	235.58	4,609	5,041	5,347	5,672
Canny HWXHWY	5.687	198.443	2,491	3,787	4,706	5,680
Canny HW2X HW2Y	2.005	565.871	942	1,374	1,680	2,005

the edge detection process execution time is decreasing with adding custom HW accelerators. However the power dissipation increases significantly. During the exploration process, depends on the design power, performance and area constraints further options can be considered and evaluated and ultimately chosen for manufacturing.

Here we generated different design architecture options based on the profiling result from initial specification model analysis. However, these design options needed to be further studied for different configuration for each design components. In [61], we proposed a systematic

and efficient multi-layer configuration framework for streaming applications. This configuration exploration framework enables rapid evaluation of MPSoCs and can be used beyond the architectural exploration for capturing efficient design.

Chapter 5

Power Optimization

Our proposed MAVO framework is designed to answer the need for system-level power and performance evaluation with minimal effort. MAVO provides thorough observability of the system-level models which assists designers to apply power and performance optimization techniques for design modification, voltage and frequency scaling, power aware scheduling, and dynamic power management with shut-off. In this chapter we investigate these optimization techniques on Canny image detector using MAVO framework.

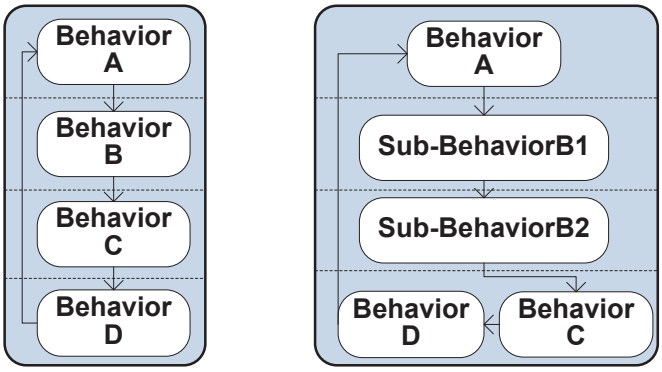
5.1 Motivation

Power and performance are major design concerns, and they directly affect all other aspect of the design, such as area, temperature, reliability and life-time of the device. However, evaluating and monitoring power and performance is a prime design challenge, particularly in multiprocessor SoCs. Therefore, a comprehensive analysis of energy dissipation within the system among HWs, communication elements, memories and SW processors is essential and can be achieved by profiling the simulation and applying power models. The features and

functionality of MAVO are designed to fill this gap, at system level. The power optimization techniques have simple idea behind them, like voltage and frequency scaling or dynamic slack reclamation, however, in order to apply the techniques either statically in design phase, or dynamically during running time, a powerful platform is required to investigate the design rapidly and with adequate details.

5.1.1 Design Modification

Multiple techniques have been proposed for optimizing power dissipation. However, a low power design is mainly efficient due to its architecture and design model itself rather than the applied optimization techniques. For instance, the effect of having a system working



(a) Original design (b) Modified design

Figure 5.1: Evaluation of different architectures for power and performance

in form of a pipeline configuration and balancing the pipeline stages, can not be made by applying a power optimization technique, such as DVFS. Figure 5.1 shows two different design options of a design. The design in Figure 5.1(a) has 4 pipeline stages, *A,B,C* and *D*. Figure 5.1(b) shows an alternative with split stage *B* (*B1* and *B2*) and merged stage *C* and *D*. Without an infrastructure to monitor and profile the performance and power in each stage, it is impossible to apply these modification and decide which architecture is more efficient.

5.2 Optimization Techniques

5.2.1 Voltage and Frequency Scaling

The fact that power is basically spending energy over time allows design optimization with respect to frequency, and supply voltage. We can reduce power dissipation and as a result develop more reliable designs by lowering the frequency or supply voltage within the defined deadline and without compromising the performance. Figure 5.2 illustrates the general idea of this scheme. The working frequency of PE_i is reduced to minimize power dissipation,

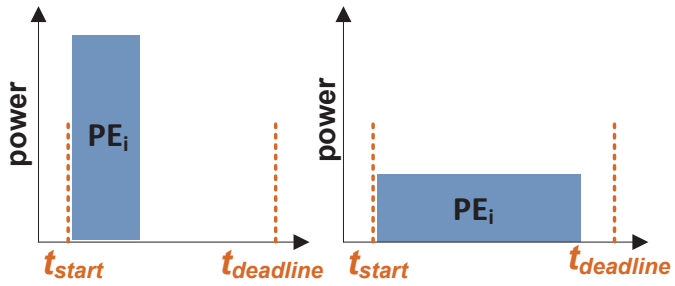


Figure 5.2: Adjusting PE clock frequency

while meeting the requested deadline.

5.2.2 Balancing Power Dissipation by Scheduling

Throughout the life-time of a device it is important to balance power dissipation. This can effectively reduce the working temperature of the device, improve reliability, minimize faults, and extend the system life-time. MAVO supports monitoring the mode and the activity intervals of each design element, as well as the amount of their power consumption. Using this information, designer can easily examine scheduling alternatives and power saving opportunities via simple, yet effective design modification. Figure 5.3 demonstrates this by

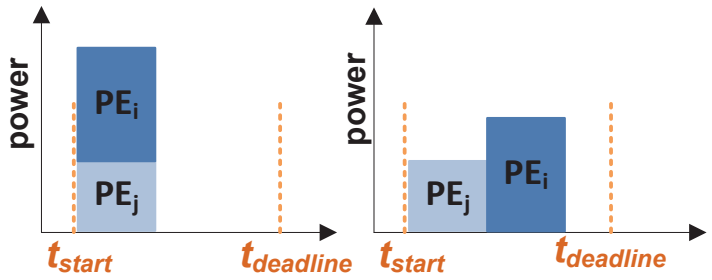


Figure 5.3: Scheduling Power Dissipation with MAVO framework

improved scheduling of the working intervals of two processing element,PE_i and PE_j, to balance the overall power usage, and reducing peaks and temperature at the same time.

5.2.3 Smoothing Power Spikes

The peak power of the design is among the factors that directly influences the reliability, thermal limitations, cost and size of the device [63]. Figure 5.4 illustrates the general idea of eliminating low and high spikes. The unwanted power dissipation behavior can be avoided by scaling frequency within the involve units. In an ideal design, peak power should be limited

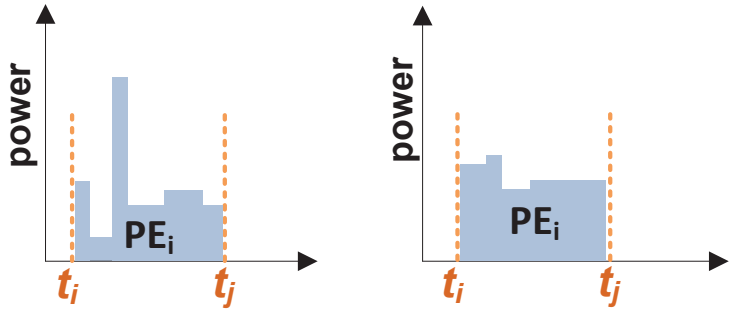


Figure 5.4: Smoothing power spikes with MAVO framework

to certain range. In MAVO we are using a simple method to monitor different active process of the design and scale down the frequency, in order to avoid out of range peak power.

5.2.4 Power Shut off

Finally, in order to reduce static power, a common Dynamic Power Management(DPM) technique is to shut off the inactive devices. MAVO also supports this approach.

5.3 Power Optimizer

The power *Optimizer* is an infrastructure for close evaluation and analysis of the design, through power reports, and identifying power and performance optimization opportunities. These opportunities can be in form of design alteration e.g. changing the weight of computations in different blocks of the design, altering algorithms, changing execution methods like parallelism or pipelining, communication policies, components allocations, and PE mappings. The other group of power saving solutions, can be power optimization methods such as dynamic voltage and frequency scaling, dynamic power management, scheduling or load balancing.

The main role of *Optimizer* is to assist the designer with optimization decisions. *Optimizer* supports generating power and performance analysis for any time interval or subsection of the design to allow the designer to evaluate the design and explore other design options rapidly. For frequency scaling, scheduling and balancing peak power, *Optimizer* can help further and show the working intervals of each element, and involved design elements in peak power.The *Optimizer* assesses PowerMeters and provides numerical logs of power over time. In order to control the size these log files and adjust the precision of this analysis, the user can pick the sampling frequency. The user can also specify any simulation interval to monitor as well. The power reports generation is an option that designer can activate through *Annotator*. The automated *Optimizer* is capable of generating power reports for

all the behaviors, processing elements, communication elements and globally. In addition to automated reports from assigned PowerMeters, user can monitor and access the report of any part of the design.

Most importantly, the user can view graphical power dissipation over time and zoom in for specific intervals of any design elements and behaviors. Moreover, the *Optimizer* support merging the reports or stacking up the power dissipation values in different PowerMeters over time. In order to optimize power, it is important to monitor all the system elements, behaviors and their interactions. For instance, a tiny change in scheduling policy or specification models result in different transactions, performance, and power dissipation. Therefore, it is a significant help to have a platform that enables analyzing the entire system thoroughly, while maintaining the speed of system level design purposes. Thanks to global, per behavior, and per elements power reports designer is able to identify further power saving solutions, examine the effect of design modification in term of power and performance, evaluate the trade-offs, and apply power optimization techniques.

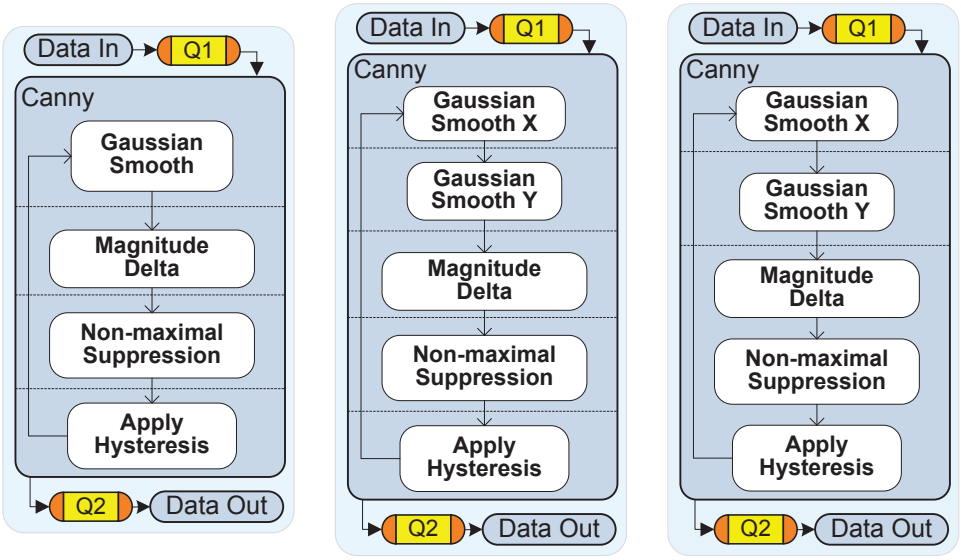
The main role of power *Optimizer* is to assist the designer to optimize power through presenting a detailed transaction of the system, communication, power related activity and performance.

5.4 Case Study: Canny Edge Detector

We have investigated the MAVO framework with a Canny edge detector. Canny is a real-life image processing application implemented in 4-stage pipeline configuration. The model was examined on an ARM based processor with two custom HW units for input and output, and 2 HWs for pipeline. All units are communicating through the AMBA BUS.

5.4.1 Canny: Design Modification

The 4-stage pipeline architecture is suggested by edge detection algorithm which has four major steps. However, after viewing the power and timing reports from MAVO, it becomes apparent that this architecture is imperfect in term of the pipeline load in each stage. Table 5.1 shows the power and time consumption of each pipeline stage. The power and timing results reveal that the *Gaussian Smooth* behavior is computationally expensive and power hungry. In order to balance the pipeline we modified the design to a 5-stage pipeline



(a) 4-stage pipeline (b) 5-stage pipeline (c) 3-stage pipeline

Figure 5.5: Canny Architecture

Table 5.1: The delay & average power of pipeline stages

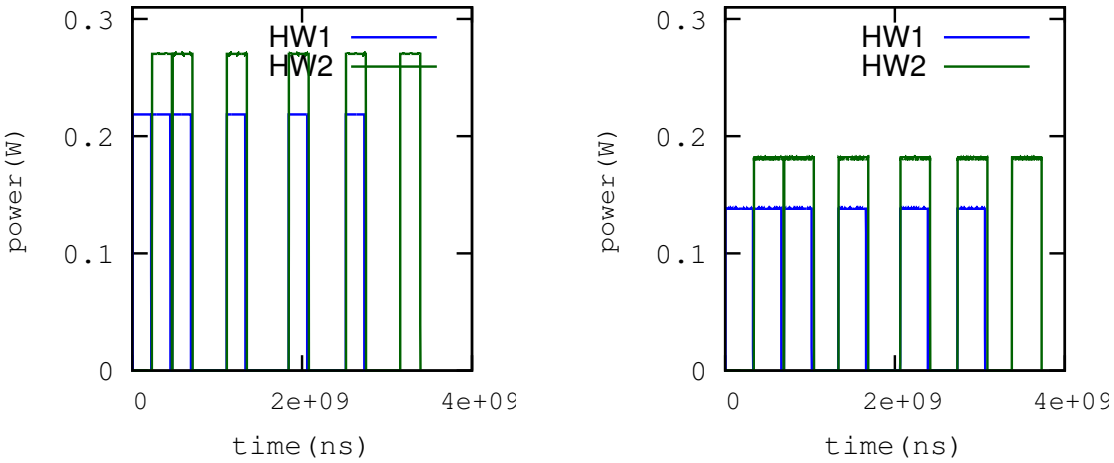
Design	Stage1		Stage2		Stage3		Stage4		Stage5	
	Time (ms)	Power (mW)	Time (ms)	Power (mW)	Time (ms)	Power (mW)	Time (ms)	Power (mW)	Time (ms)	Power (mW)
4-Stage	537	328.3	184.4	77.1	353	72.5	142	38.5	-	-
5-Stage	226	174.6	237.8	149.6	184.4	77.1	353	72.5	142	38.5
3-Stage	226	174.6	237.8	149.6	688.8	188.2	-	-	-	-

(CannyA), splitting the Guassian Smooth behavior in to X and Y dimensions. In this work

the stage 1 and stage 2 of the pipeline are mapped to custom HWs, and rest of the stages are mapped to ARM processor. The applied mapping makes the Canny architecture works as a 3-stage pipeline configuration. Figure 5.5 shows the architecture of Canny before and after the modification. Next we evaluate canny for power optimization.

5.4.2 Canny: Adjusting Frequency

Using the reports and graphs, the working frequency and supply voltage of each unit can be optimized. In Figure 5.10(a), a power saving opportunity can be detected for HW1 and HW2, which finish their tasks earlier than the rest of stages. In turn, we lower the frequency of HW1 and HW2 within the performance constraints (CannyB).



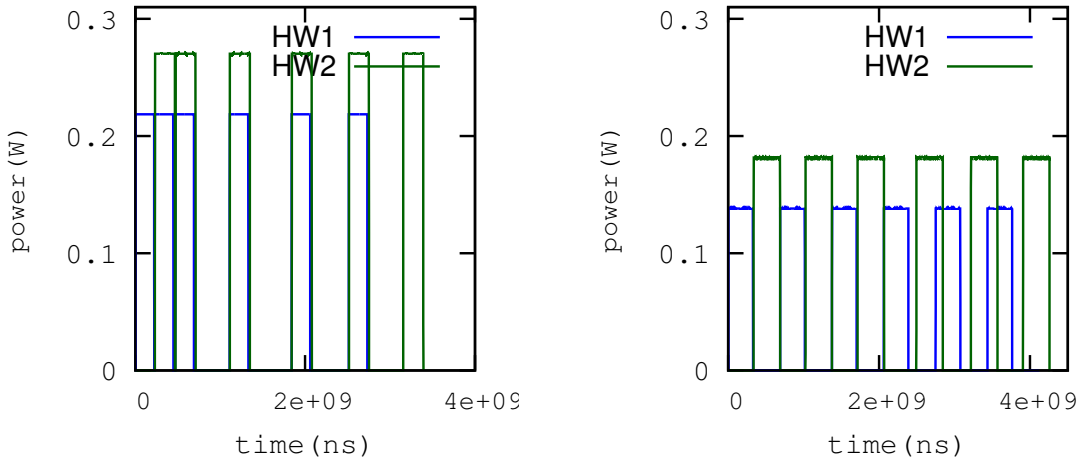
(a) Before (b) After

Figure 5.6: Adjusting Frequency for HW1 and HW2 using MAVO

By extending the processing time in stage 1 and 2, the simulation time gets extended as well. This is due to the fact that filling the pipeline stages takes longer, however, the pipeline throughput performance remains the same.

5.4.3 Canny: Power Aware Scheduling

In order to balance power dissipation in whole device, we can schedule the work period of the units such that they have minimum overlaps. For HW1 and HW2 the result of this modification (CannyC) is shown in Figure 5.7.



(a) Before (b) After

Figure 5.7: Adjusting work period for HW1 and HW2 using MAVO

5.4.4 Canny: Smoothing Power Spikes

The total power results from MAVO show that the device is experiencing high peak powers, where the peak is more than the double the average.

In order to smoothen the dissipation MAVO identifies the active processes during the peaks as shown in Figure 5.8. Here we decide to scale the frequency for the involved behaviors. Figure 5.9 shows the results (CannyD).

A block performing the floating point operations is responsible for power peaks. We lower the

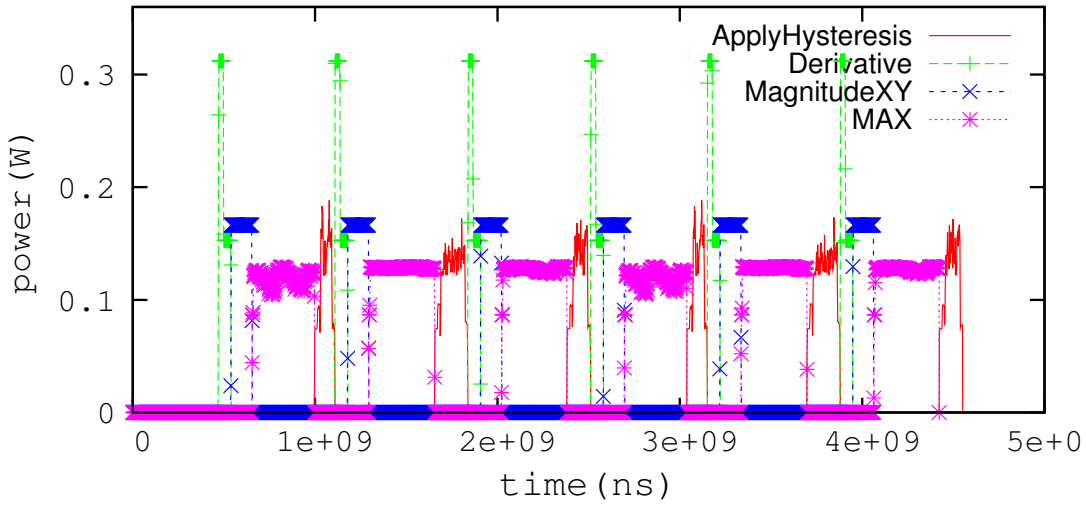
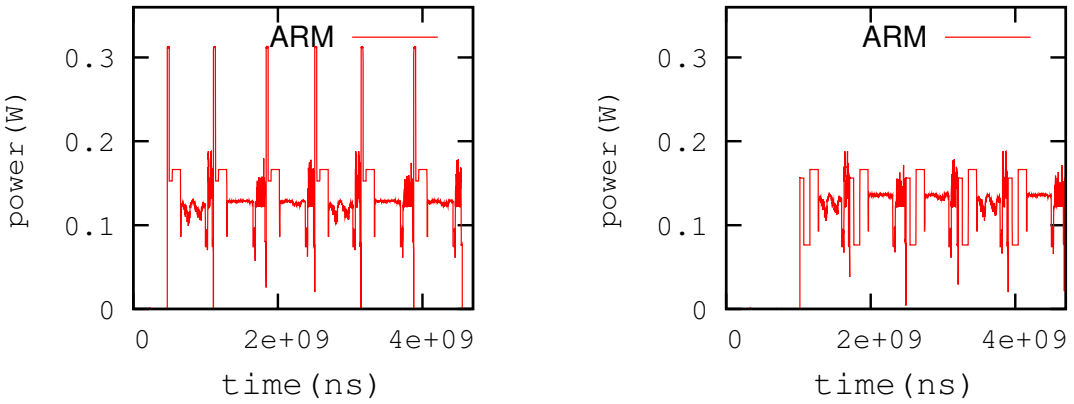


Figure 5.8: Active processes power dissipation in CPU



(a) Before

(b) After

Figure 5.9: Smoothing power dissipation using MAVO

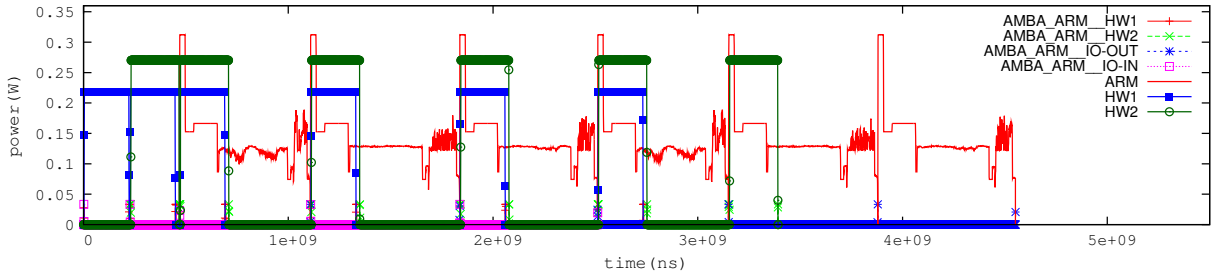
frequency of CPU here and in order to maintain the performance, another integer intensive behavior is scaled with higher frequency instead.

5.4.5 Results

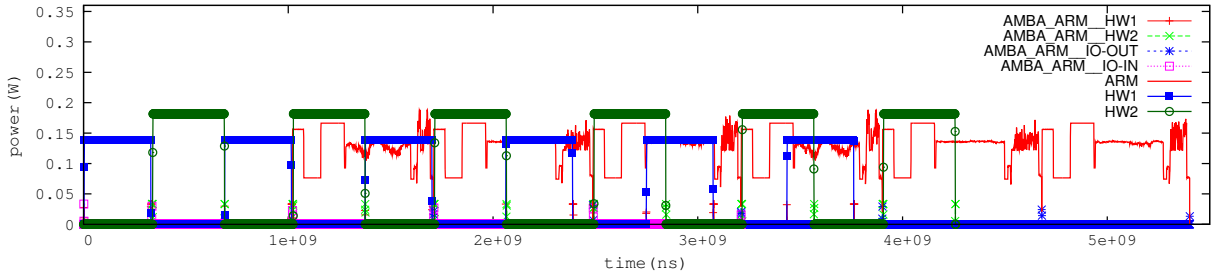
Table 6.1 shows the power and performance of each models. The canny example has been tested with 6 images and it was expected to generated one image in every 0.8 seconds. As shown, MAVO power savings resulted in an optimized design with no performance penalty. The optimized design experiences power fluctuations 8% less, based on comparing the standard deviation of power reports and the power changes range has been reduced by 29%. The difference between the minimum and the maximum of power dissipation over time, considered as the power changes range of the design.

Table 5.2: Timing and performance for Canny edge detector after applying each technique

Model	Pipeline Throughput (ms)	Power Range (min,max)	Relative Fluctuation	Power (mW)	Power Saving
CannyA	688.8/800 (86%)	(0.003,0.312)	100%	374.504	+0%
CannyB	689/800 (86%)	(0.003,0.312)	100%	357.113	+5%
CannyC	701.8/800 (88%)	(0.003,0.312)	100%	329.903	+12%
CannyD	738.8/800 (92%)	(0.001,0.204)	71%	315.511	+16%



(a) Canny edge detector



(b) Optimized Canny Edge Detector

Figure 5.10: Power dissipation of Canny Edge Detector visualized and optimized by MAVO

Chapter 6

Static Analysis of Power and Performance

In this chapter we present a novel abstract verification approach for evaluating power and performance constraints and power management. The proposed design flow combines a method for generating power- and performance-aware ESL models presented in MAVO with a technique to extract a corresponding UPPAAL model from the ESL model in [24].
A hierarchical concurrent automata model with integrated power and performance information is generated which can be statically verified against various design constraints using the UPPAAL model checker.

To explore the power and performance trade-off we present an algorithm that efficiently navigates and determines the trade-off curve. In the experimental results we examine the implemented power/performance framework using two JPEG image encoders.

6.1 Motivation

The embedded system design process is getting more and more complicated. Design functionality verification is not the major concern any more. Power and performance constraint evaluations along with design optimization are the main challenges nowadays. In order to improve the design process as well as reducing the time-to-market, the aspects of design analysis are start at the Electronic System Level (ESL).

There are many solutions and commercial tools for power and performance estimation, where the common technique among all of them is using simulation as the basic approach for design space exploration and metrics analysis. The simulation approach allows the designer to evaluate different design alternatives, trace timing and power in different design components, and basically extract any required information. However, due to long simulation times only a subset of the cases can be examined and this does not guarantee safe boundaries for power and performance, particularly for real-time and power sensitive applications. Therefore a simulation approach degrades the reliability of the evaluations.

Another solution for design constraints investigations are formal methods. There are less studies on formal methods for power and performance verification and optimization. The model-based static analysis allows evaluation of system level models against different design constraints, design optimization options, component alterations, and generally any design decisions. The downside with formal models is the systematic generation of the automata models from a system level model, besides the availability of model checkers to evaluate the models efficiently. Since system models are conventionally created in System Level Description Languages (SLDL), an additional step is required to generate formal models. In this work we are proposing a framework for static analysis of power and performance using the UPPAAL [11] model checker. Our main contributions of this work are:

- We extended [71], a power and performance ESL estimation tool to generate annotated

ESL models with different level of granularity.

- We designed a model generator base on [24] for automated conversion of power- and performance- aware ESL models to a power-timed automata network with integrated voltage and frequency scaling technique.
- We proposed an algorithm to automatically explore the power and performance trade-off using the UPPAAL model-checker.

To the best of our knowledge this is the first proposed framework for static analysis of power and performance of ESL models.

6.2 Related Work on Formal ESL Model-Checking

Although formal verification is rarely used for power and performance analysis, it has been broadly applied in deadlock/livelock and parallelism analysis, as well as race condition detection. [74], [75], [13] are proposed to detect races and analyze non-deterministic anomalies in timed concurrent models, but in those methods simulation is required. As for the system modeling, [80] and [12] propose approaches to model the behavior of SLDL designs with automata in PROMELA, and in [44] the design is modeled as a network of timed automata. The model is then analyzed with SPIN and UPPAAL model checker, respectively. In [28] model-checking is applied for performance analysis using UPPAAL model checker. Our approach also models the application with a network of automata and analyzes it with UPPAAL model checker. Compared with other works, our approach supports the modeling of richer design compositions and channel communication. More important in our proposed approach is that the power and performance trade-off can be statically evaluated, and power management functionality in form of voltage/frequency scaling is automatically integrated to ESL models. In the UPPAAL model generator in [23], the regular finite state automata are

equipped with time, so called timed automata, through a variable for representing the clock. In this work we extended the timed automata with the dimension of power by introducing extra static and dynamic energy dissipation variables.

6.3 The Static Analyzer Framework

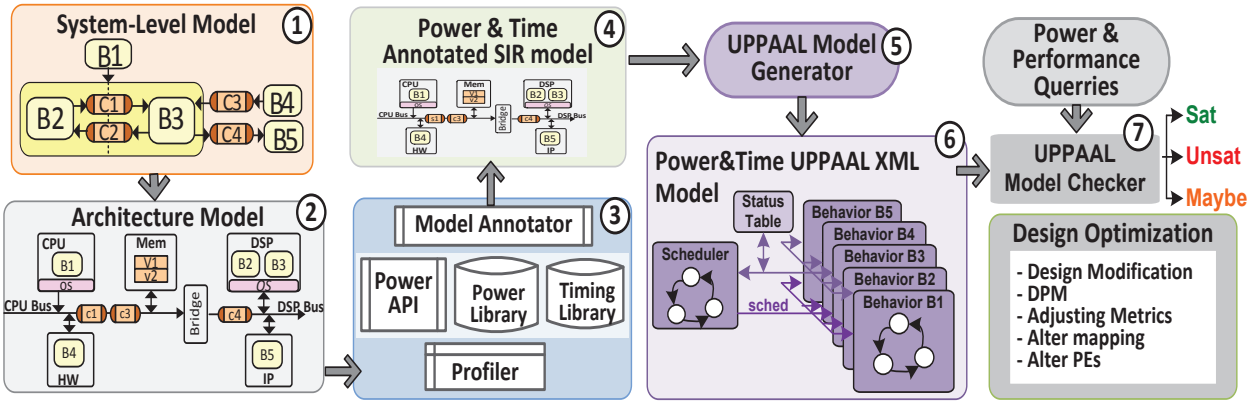


Figure 6.1: The Static Power and Performance Analyzer Framework Flow

Figure 6.1 illustrates the proposed static analyzer framework. Our framework is composed of two major units; a power and performance estimation tool that generates a Power- and Performance-Aware (PPA) model from an ESL architectural model in steps (1) through (4), and a UPPAAL model generator that transforms PPA models to a formal automata network that matches UPPAAL model checker semantics, steps (5) and (6).

The design process starts with a system level specification model written in SLDL. The ESL model contains a well-defined structural hierarchy, separate computation and communication, and explicit parallelism. In this work, we implemented the system level specification model in SpecC language[34], step (1). The design components allocation, mappings, and refinements are performed using the System-on-Chip Environment [26], step (2). At step (3) the refined architecture model is instrumented with power and performance details, to

generate power-timed models at step (4). These four steps are similar to first four steps of MAVO framework presented in Chapter 2.

In the next two steps, the annotated ESL model is converted to UPPAAL model as shown in Figure 6.1. The UPPAAL model checker examines the power-timed automata network model against different queries, step (7).

6.3.1 Power and Performance Estimator Tool

In [71] a framework for monitoring and analyzing power and performance at the system level is presented. In our work we have adjusted this to monitor and annotate the ESL architecture model with thorough traces on power and performance, that are compatible with our designed UPPAAL model generator.

As shown in Figure 6.1, the design process at system level starts with a specification model, that reflects the functional behavior of the system, without any notion of time nor power. Next, Processing Element (PE) allocation, behavior mappings, scheduling and communication refinements are performed in SCE [26]. The [71] framework uses a *Profiler*, a *Power API* and a model *Annotator* to generate PPA models. It initially monitors the architecture model through profiling of different operations executed on processing elements as well as memory accesses, besides the amount and type of data being transferred over the channels in the model.

In the *Power API*, a set of power and timing related functions are provided to add the dimensions of power and time to ESL models. Using the provided functions and power models of design components, [71] specifies power and performance related activities and analyzes power and performance dissipation in different processing elements, communication elements and behaviors automatically using the *Annotator*. The collected traces from the *Profiler* are instrumented to *Power API* functions then back-annotated to the system level model via *Annotator*. In this work, we have extended the *Annotator* capabilities to instrument the

power and performance annotations at different levels of granularity such as basic blocks, behaviors, or PEs. We have also extended the *Power API* function set, to support power and performance trace integration for the UPPAAL model generator. An example of a behavior annotated with time and power information is presented below.

```
Behavior A(i_int_sender Ch)
{
 note A._MAPPED_TO = "ARM968E–S";
 ...
 d++;
 Ch.send(d);
 {
  UPPAAL:
    pm_dissipate(2.3 NANO_SEC, 3.47 MILLI_JOULE, 2.84 MILLI_JOULE);
 }
};
```

As shown in the above example, the *dissipate* function represents the spent time as well as the power due to dynamic and static power dissipation in *Behavior A* which is mapped to $ARM968E-S$ with behavior level selected as the granularity of annotations.
The applied annotations contain the same information about performance and static and dynamic power as the annotations in MAVO framework presented in Chapter 2. However, naming convention is modified to adapt with UPPAAL model generator which embeds this information into automata of the system-level model. This process is explained in the next section.

6.3.2 UPPAAL Model Generator

An UPPAAL model consists of a network of concurrent processes and the network is created by instantiating automaton templates. Step (7) in Figure 6.1 illustrates the structure of the

UPPAAL model for a system model. The ESL model composed of multiple computation blocks (modules and behaviors), with communication (port, channel, event synchronization) in between is converted to an automaton template for each behavior [24]. Every instance of a behavior is mapped to an UPPAAL process.

Our model generator supports different orders of execution, such as sequential, pipeline, FSM, and parallel. Control flow statements (if/if-else, while/do-while, for loop), synchronization statements (*wait*, *waitfor*) and channel communications (semaphore, mutex, handshake, double-handshake, and queue) are also incorporated into the UPPAAL templates. The system also contains a scheduler process to coordinate the transitions in instance processes via connections to the scheduler process.

Our UPPAAL model generator explicitly supports power and performance annotations from PPA models. Our extended *Annotator* [71] offers different granularity of power and performance instrumentation, from basic block, to behaviors and design components. Thus, the UPPAAL model generator is able to capture the instrumentation various granularity levels similarly. An example of capturing instrumented power and performance details from power- and performance-aware model at behavior level is shown in Figure 6.2. For each

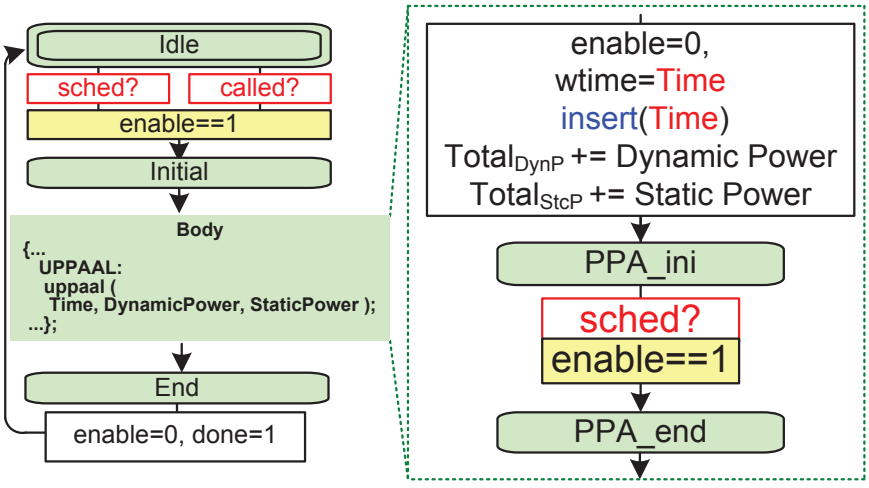


Figure 6.2: Representation of power and performance annotation of a behavior in UPPAAL

annotated *dissipate* function, two *locations* (states) $[PPA\_ini]$ and $[PPA\_end]$ are created.

The dynamic and static power are also inserted as *labels* (an operation during transition). The annotated time value is treated similar to *waitfor* statements in simulator, where it suspends the current instance from execution for the specified time units.The suspended process is reactivated by the scheduler after the simulation time is advanced. As shown in Figure 6.2, $Total_{DynP}$ and $Total_{StcP}$ are two variables introduced in UPPAAL models to capture the dynamic and static power consumption.

6.4 UPPAAL Model-Checker

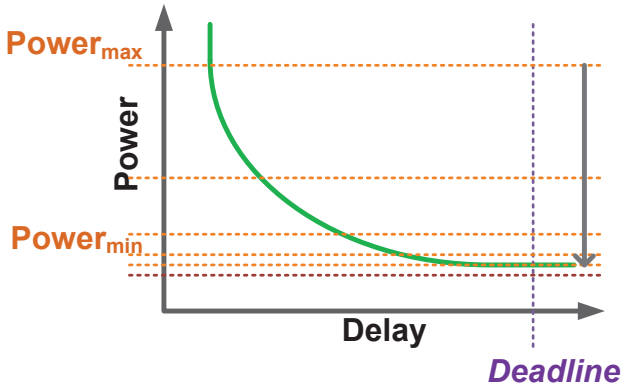
Once the UPPAAL model is generated, different design criteria, such as power and performance, can be verified within the whole system design model or its subsets, such as behaviors. In other words, the verification can be applied on the network of power-time automata or on any subset of automata.

The evaluation is performed through queries to the UPPAAL model-checker. For instance, $Query_A$*(Scheduler.Terminate,* P_i*,* T_j*)* can be used to examine if the UPPAAL model of an application A, will be satisfied with property of total power and delay less than P_i and T_j respectively. The term *Scheduler.Terminate* represent the scheduler automata to be in its *terminate* state. This implies that the property is checked for the application A for its entire life-time.

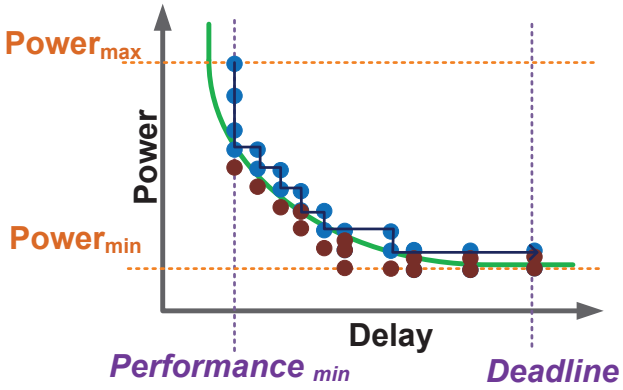
Although different verification can easily be tested using model-checker, in order to systematically explore the design space, and find the power-performance curve of the design, rather than exhaustive verification, we present an algorithm in Section 6.5.

6.5 Power and Performance Trade-off

In order to explore the power and performance space of the system level model, we propose an algorithm to statically analyze the model through repeated queries for power and performance values. Our algorithm allows the designer to rapidly evaluate the constraints of the generated model. In our method, the designer only needs to provide the deadline and



(a) Finding the minimum Power



(b) Determining the power-performance trade-off curve

Figure 6.3: Power & performance trade-off

maximum power. Figure 6.3 illustrates two steps of the algorithm. In the first step the minimum possible power and performance is found by queries generated based on binary search, as shown in Figure 6.3(a) for power. Using the similar approach the minimum value for delay is found as well. Next, to find the optimal power and performance combination

for the ranges of interest, we propose Algorithm 1. This Algorithm starts searching the area

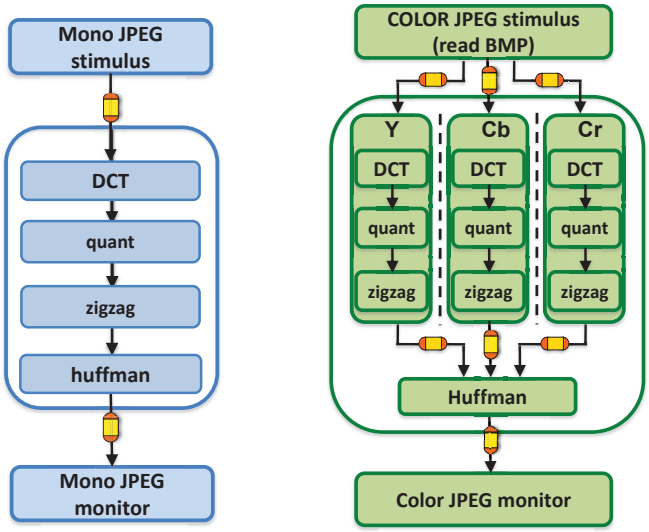
Algorithm 1 Query Generation Algorithm for Finding the Power and Performance Trade-off

```
1:  let the Power_max be the maximum allowed power of the design
2:  let the Deadline be the deadline of the design
3:
4:  void GeneratePowerPerformanceTradeOff(Power_max, Deadline)
5:  {
6:    Power_min = FindSatisfiableMin(power, 0, Power_max);
7:    Performance_min = FindSatisfiableMin(performance, 0, Deadline);
8:    Initialize currentPerformance to Performance_min
9:    Initialize currentPower to Power_max
10:   while( currentPerformance<Deadline){
11:     while( currentPower<Power_min){
12:       check = Query(Scheduler.Terminate, currentPower, currentPerformance);
13:       if( check == Satisfiable) then
14:         set currentPower to currentPower + Power Step;
15:       else if (check != Satisfiable and currentPower within Range) then
16:         set currentPower to (currentPower + Last Satisfiable Power Value)/2;
17:       else break;
18:       end if
19:       Trade-Off(currentPerformance, currentPower) = check;
20:     end while
21:     currentPower) = Last Satisfiable Power Value;
22:     if (currentPower) < Lowest Satisfiable Found Power Value) then
23:       set currentPerformance to currentPerformance + Time Step;
24:     else
25:       Double the Step Size;
26:       set currentPerformance to currentPerformance + Time Step;
27:     end if
28:   end while
29:   }
30: }
31:
32: double FindSatisfiableMin(property,min,max){
33:   if(absolute(max-min)<Range) then
34:     return max;
35:   end if
36:   middle=(min+max)/2;
37:   check = Query(Scheduler.Terminate, property<middle);
38:   if(check == Satisfiable) then return FindSatisfiableMin(middle,max);
39:   else return FindSatisfiableMin(min,middle); end if
40: }
```

of interest from the maximum power and minimum delay point (top left in Figure 6.3(b)). As presented in the algorithm, there are two main processes. In the inner loop the lowest satisfiable power for every value of performance is found. Then based on the determined power value, the performance is increased accordingly and the same process is repeated. The resolution of the exploration is a parameter that can be set by the designer.

6.6 Experimental Results

To evaluate the proposed approach, we have extended the power and performance estimator tool [71], and power-timed automata generation [24] compatible with the UPPAAL model-checker. The power and performance trade-off algorithm is also implemented and tested. We have tested a grayscale and a color JPEG image encoders with our framework. The mono



(a) Grayscale JPEG block diagram

(b) Color JPEG block Diagram

Figure 6.4: JPEG Block Diagram

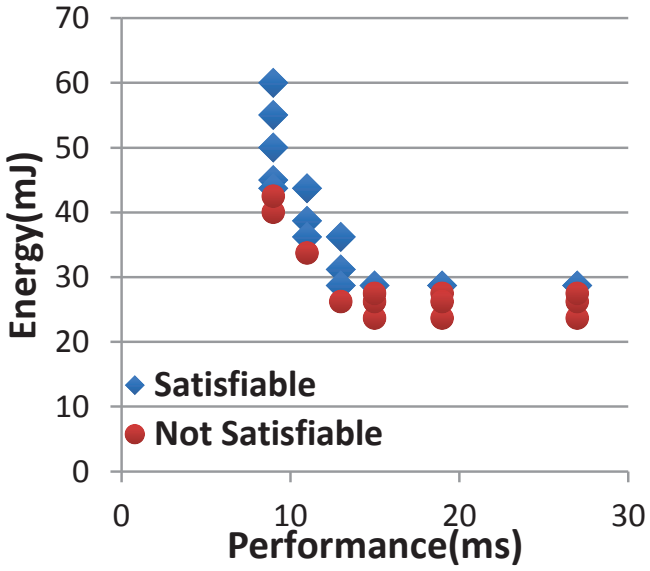
Table 6.1: Experimental result for JPEG applications

Embedded Application	Lines of Code	Analysis Time Satisfiable (sec)		Analysis Time Not Satisfiable (sec)		Peak Memory	Number of Satisfiable Query	Number of Not Satisfiable Query	Total Time (min)
		Min.	Max.	Min.	Max.				
Mono JPEG	1.5k	0.20	2.44	37.2	28.6	1,417MB	14	13	10:27
Color JPEG	2.5k	0.01	0.87	22.3	23.89	1,100MB	20	16	6:23

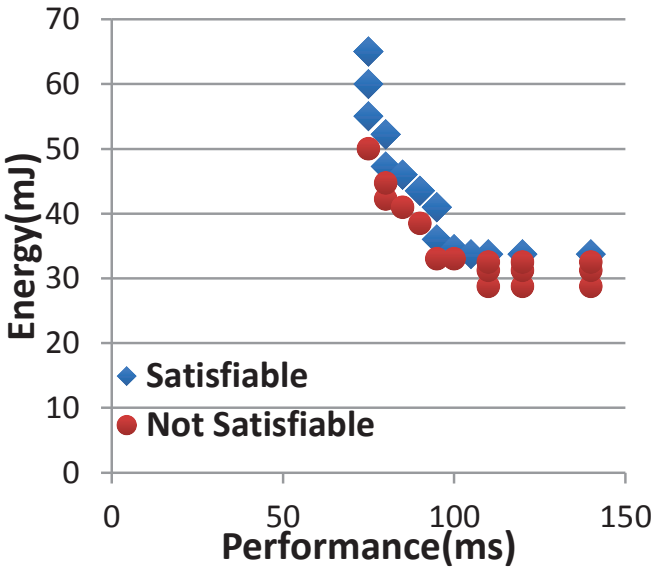
JPEG, Figure 6.5(b), is mapped to an ARM-based processor with a custom HW for DCT block. The color JPEG image encoder shown in Figure 6.5(a) performs DCT, quantization, and zigzag operations in parallel followed by a Huffman encoder. In our experiments, the three color components are mapped to custom HWs and the other components are mapped

to an ARM based processor. All the units are communicating through an AMBA bus. The automated process of generating power- and performance-aware model and UPPAAL model took *27* seconds for grayscale and *40* seconds for color JPEG application. Then the resulting power-timed automata network is studied using the UPPAAL model checker. The proposed algorithm for power-performance trade-off is utilized to compute power and performance values for queries. The *Time Step* is set to *5ms* for grayscale JPEG and *2ms* for color JPEG in Algorithm1. Figure 6.5 shows the calculated trade-off curves for the two color JPEG encoders. As shown, the optimal design options for power and performance values are rapidly examined though a very limited number of property checking, less than *30* and *40* queries for the mono and color JPEG encoder respectively.

Although in our experiments, we are presenting the result for global power-performance trade-off, other properties such as power and performance of a subset of behaviors, static or dynamic power dissipation, timing bounds, deadlock, etc. can also be verified through queries.



(a) Grayescale JPEG Encoder



(b) Color JPEG Encoder

Figure 6.5: Determine Power & performance trade-off for (a) Mono JPEG and (b) Color JPEG Encoder

Chapter 7

Conclusion

The chip design is evolving day by day in response to semiconductor technology size. The reduction in technology size as Moores's law predicted caused an exponential increase in chip density and introduced complex chips, SoCs and MPSoCs. These progresses has challenged the design methodologies and poses new design concerns such as power, temperature and reliability.

Design methodologies are applying various approaches such as design abstraction, language-based design and synthesis, and electronic design automation to increase the designer productivity. On of the primary solution for tackling design complexity and hardware and software design gap is system-level design. The high level of abstraction, higher potential for power, area and performance optimization, rapid design space exploration and design modification has made this level a popular and great starting point for SoC design. Therefore, different design instruments such as design languages, simulators, design verification ans evaluation tool, etc. should get developed or adapted for more effective system-level design.

The deep submicron technology advances allows implementing tens of millions of gates in a very small die, however, different design problem needs to be addressed and resolved, such as high power dissipation, temperature, high leakage current, which can get as high as dynamic

current. Therefore new design solutions such as using multi-processor on the chip instead of a mono-processor are proposed.

In this work we are targeting power estimation and evaluation as a major SoC design concern in system-level design. We proposed a framework for monitoring design specification models for power activity and evaluate power dissipation in different design components and with different granularity levels.

7.1 Contributions

We summarize our contributions in the following sections.

7.1.1 A System Level Power Estimator

In Chapter 2 with presented the MAVO framework for power estimation at system level. The MAVO frameworks is an automated and simulation-based power estimator for system level design. The main modules in MAVO framework are:

- **Monitor**: For profiling the specification model for execution counts of different operations an statements, communication, and required memory via simulation. The generated static report of Monitor is then used for producing a power-annotated design model.
- **Power Analyzer API** : This API is presented in form of a library for instrumenting power related information to the design. PowerMeter structure, monitor, and consume functions are developed in Power Analyzer API to provide capabilities for monitoring power over time and capturing total power dissipation and power spikes in different level of granularity and design elements.

- **Annotator**: To eliminate the manual effort and provide a scalable power estimator tool an Annotator is implemented for annotating PowerMeters and consume functions to the design at different granularity levels.

These three modules are collaborating to produce a power-annotated system level model. The text and graphical power reports for different design components are generated after simulating the power-annotated model. A real-word application, JPEG encoder is used to demonstrate capabilities and provides power-time graph reports.
In Chapter 3, we verified the fidelity and relative accuracy of the MAVO framework. We first presented a simplified power modeling scheme to generate power library of different design components such as processors and custom hardwares. Then, using a real-life applications, JPEG encoder, MP3 encoder, and H.264 codec we investigated the accuracy and fidelity. Our experimental results demonstrate the strong fidelity and an order of magnitude speedup using MAVO in compare to cycle-accurate power estimator tools.

7.1.2 Power-Aware Design Space Exploration

The desired accuracy for design constraints evaluation at system level is fidelity. The fact that the MAVO power estimator support high degree of fidelity in power evaluations motivates us to add power as a new dimension into our in house design space exploration tool, SCE. Hence, we integrated the MAVO modules into SCE and extended its database with power models. Chapter 4 gives an overview of the tool and describes the exploration flow using the Canny edge detector application.

7.1.3 Interactive Power Optimization Support

The power dissipation in SoC can be optimized in all design abstraction levels, however the in lower levels there are limited margin high cost for power reduction and design alteration respectively. This had made the system-level a great level for starting power optimization and considering power in design alterations. In order to evaluate power and apply power optimization techniques, first a comprehensive understanding of the design model, along with power reports of power dissipation within design components and behaviors is essential. MAVO has an easy solution to collected these information and examine the applied solution. The generated power profile reports helps the design to understand the power dissipation behavior of the design for utilizing power optimization technique or changing the design accordingly.

In Chapter 5 we used Canny edge detection application for power optimization within MAVO. Our results show a great reduction in total power consumption and flattening of power spikes.

7.1.4 A Platform for Static Analysis of Power and Performance

Due to the fact that capturing the power-performance evaluation for all design options needs iterative simulation and therefore is it a time consuming task, we proposed a verification approach for evaluating power and performance constraint and power management using static analysis. In Chapter 6 we proposed a design flow that combines a modified MAVO with an extended automata model generator to evaluate the design constraints through different queries in UPPAAL model checker. To explore the power and performance trade-off we present an algorithm that efficiently navigates and determines the trade-off curve. In the experimental results we examine the implemented power/performance framework using two JPEG image encoders.

7.2 Future Work

Related future research work is worth pursuing in the following areas:

7.2.1 Automated Power Optimization

For future work we are planning to deliver power optimization techniques to be applied automatically at the system-level. As of now our proposed infrastructure only supports the interactive optimization which is user-based.

7.2.2 Extension for Reliability Analyzer

Also we will address integrating an efficient dynamic power manager with thermal and reliability analyzer. Research study is also needed to investigate reliability, aging and variation in full chip, which some research such as [43] has already proposed new design methodologies that considered these design criterion as well.

7.2.3 Efficient Static Analyzer for Power-Performance

In the future, we will focus on supporting different power optimization techniques and try to simplify the complexity of the automata networks. We aill also try to adapt the automata generation to SystemC language, which is the industry standard for system-level design.

7.3 Concluding Remark

In conclusion, the work presented in dissertation provide a rapid system-level power estimator tool. MAVO is framework suitable for automated power estimation, design space exploration, and investigating power optimization solutions interactively. MAVO also support statical analysis of power-performance trade-off together with UPPAAL framework. Overall, MAVO is an environment for low power embedded system design that eventually can lead to a significantly shorter design cycle.

Bibliography

[1] Design Compiler, Synopsys Inc., 2000.

[2] Open SystemC Initiative, http://www.systemc.org. Functional Specification for SystemC 2.0, 2000.

[3] IEEE Standard Verilog Hardware Description Language. IEEE Standard 1364-2001, 2001.

[4] IEEE Standard VHDL Language Reference Manual. IEEE Standard 1076-2008 (Revision of IEEE Standard 1076-2002), 2009.

[5] PowerPlay Early Power Estimator, Altera Inc., 2009.

[6] Hadi S Aghdasi, Maghsoud Abbaspour, Mohsen Ebrahimi Moghadam, and Yasaman Samei. An energy-efficient and high-quality video transmission architecture in wireless video-based sensor networks. *Sensors*, 8(8):4529–4559, 2008.

[7] Sumit Ahuja, Avinash Lakshminarayana, and Sandeep Kumar Shukla. *Low Power Design with High-level Power Estimation and Power-aware Synthesis*. Springer, 2011.

[8] Takuya Azumi, Yasaman Samei Syahkal, Yuko Hara-Azumi, Hiroshi Oyama, and Rainer Dömer. TECSCE: HW/SW Codesign Framework for Data Parallelism Based on Software Component. In *Embedded Systems: Design, Analysis and Verification*, pages 1–13. Springer, 2013.

[9] Felice Balarin. *Hardware-software co-design of embedded systems: the POLIS approach*. Springer, 1997.

[10] Nikhil Bansal, Kanishka Lahiri, Anand Raghunathan, and Srimat T Chakradhar. Power Monitors: A framework for system-level power estimation using heterogeneous power models. In *VLSI Design, 2005. 18th International Conference on*, pages 579–585. IEEE, 2005.

[11] Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, John Hakansson, Paul Petterson, Wang Yi, and Martijn Hendriks. UPPAAL 4.0. In *Quantitative Evaluation of Systems, 2006. QEST 2006. Third International Conference on*, pages 125–126. IEEE, 2006.

[12] Béatrice Bérard, Michel Bidoit, Alain Finkel, François Laroussinie, Antoine Petit, Laure Petrucci, and Philippe Schnoebelen. *Systems and software verification: model-checking techniques and tools*. Springer Publishing Company, Incorporated, 2010.

[13] Nicolas Blanc and Daniel Kroening. Race analysis for SystemC using model checking. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 15(3):21, 2010.

[14] David Brooks, Vivek Tiwari, and Margaret Martonosi. *Wattch: a framework for architectural-level power analysis and optimizations*, volume 28. ACM, 2000.

[15] Doug Burger and Todd M Austin. The SimpleScalar tool set, version 2.0. *ACM SIGARCH Computer Architecture News*, 25(3):13–25, 1997.

[16] Lukai Cai and Daniel Gajski. Introduction of design-oriented profiler of SpecC language. *Center for Embedded Computer System*, 2002.

[17] Lukai Cai, Junyu Peng, Chun Chang, Andreas Gerstlauer, Hongxing Li, Anand Selka,

Chuck Siska, Lingling Sun, Shuqing Zhao, and Daniel D Gajski. *Design of a JPEG encoding system*. Citeseer, 1999.

[18] Canny Edge Detector. ftp://figment.csee.usf.edu/pub/Edge_Comparison/source_code/canny.src.

[19] Trevor E Carlson, Wim Heirman, and Lieven Eeckhout. Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 52. ACM, 2011.

[20] R Chadha and J Bhasker. An ASIC Low Power Primer, 2013.

[21] Pramod Chandraiah and Rainer Dömer. *Specification and design of a MP3 audio decoder*. PhD thesis, Citeseer, 2005.

[22] Che-Wei Chang and Rainer Dömer. System Level Modeling of a H. 264 Video Encoder. *Center for Embedded Computer Systems*, 2011.

[23] Che-Wei Chang and Rainer Dömer. *Abstracting ESL Designs to UPPAAL System Models*. PhD thesis, Citeseer, 2014.

[24] Che-Wei Chang and Rainer Dömer. May-Happen-in-Parallel Analysis based on Segment Graphs for Safe ESL Models. In *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2014.

[25] International Roadmap Committee et al. International Technology Roadmap for Semiconductors, 2011 Edition. *Semiconductor Industry Association, http://www. itrs. net/Links/2011ITRS/2011Chapters/2011ExecSum. pdf*, 2011.

[26] Rainer Dömer, Andreas Gerstlauer, Junyu Peng, Dongwan Shin, Lukai Cai, Haobo Yu, Samar Abdi, Daniel D Gajski, et al. System-on-chip environment: a SpecC-based

framework for heterogeneous MPSoC design. *EURASIP Journal on Embedded Systems*, 2008.

[27] Xiangyu Dong, Cong Xu, Yuan Xie, and Norman P Jouppi. Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 31(7):994–1007, 2012.

[28] Maher Fakih, Kim Grüttner, Martin Fränzle, and Achim Rettberg. Towards performance analysis of SDFGs mapped to shared-bus architectures using model-checking. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1167–1172. EDA Consortium, 2013.

[29] Bernhard Fischer, Christian Cech, and Hannes Muhr. Power modeling and analysis in early design phases. In *Proceedings of the conference on Design, Automation & Test in Europe*, page 197. European Design and Automation Association, 2014.

[30] Daniel D Gajski, Samar Abdi, Andreas Gerstlauer, and Gunar Schirner. *mbedded System Design: Modeling, Synthesis and Verification.* springer, 2009.

[31] Daniel D Gajski and Robert H. Kuhn. Guest editors' introduction: New VLSI tools. *Computer*, 16(12):11–14, 1983.

[32] Daniel D Gajski, Sanjiv Narayan, Loganath Ramachandran, Frank Vahid, and Peter Fung. System design methodologies: aiming at the 100 h design cycle. *IEEE Transactions on Very Large Scale Integration(VLSI) Systems*, 4(1):70–82, 1996.

[33] Daniel D Gajski, Frank Vahid, Sanjiv Narayan, and Jie Gong. Specification and design of embedded systems. 1994.

[34] Daniel D Gajski, Jianwen Zhu, Rainer Domer, Andreas Gerstlauer, and Shuqing Zhao. SpecC: Specification Language and Methodology. 2000.

[35] Andreas Gerstlauer, Rainer Dömer, Junyu Peng, and Daniel D. Gajski. *System Design: A Practical Guide with SpecC*. Kluwer Academic Publishers, 2001.

[36] Andreas Gerstlauer, Christian Haubelt, Andy D Pimentel, Todor P Stefanov, Daniel D Gajski, and Jürgen Teich. Electronic System-Level synthesis Methodologies. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 28(10):1517–1530, 2009.

[37] Marco Giammarini, Massimo Conti, and Simone Orcioni. System-level energy estimation with Powersim. In *Electronics, Circuits and Systems (ICECS), 2011 18th IEEE International Conference on*, pages 723–726. IEEE, 2011.

[38] Miltos D Grammatikakis, George Kornaros, and Marcello Coppola. Power-Aware Multi-core SoC and NoC Design. In *Multiprocessor System-on-Chip*, pages 167–193. Springer, 2011.

[39] David Greaves and Mehboob Yasin. TLM POWER3: Power estimation methodology for SystemC TLM 2.0. In *Models, Methods, and Tools for Complex Chip Design*, pages 53–68. Springer, 2014.

[40] Kim Grüttner, Philipp A Hartmann, Kai Hylla, Sven Rosinger, Wolfgang Nebel, Fernando Herrera, Eugenio Villar, Carlo Brandolese, William Fornaciari, Gianluca Palermo, et al. The COMPLEX reference framework for HW/SW co-design and power management supporting platform-based design-space exploration. *Microprocessors and Microsystems*, 37(8):966–980, 2013.

[41] Xu Han, Weiwei Chen, and Rainer Doemer. A Parallel Transaction-Level Model of H. 264 Video Decoder. *Center for Embedded Computer Systems*, 2011.

[42] Xu Han, Yasaman Samei, and Rainer Doemer. System-level modeling and refinement of a canny edge detector. *Center for Embedded Computer Systems*, 2012.

[43] Domenik Helms, Kim Gruttner, Reef Eilers, Malte Metzdorf, Kai Hylla, Frank Poppen, and Wolfgang Nebel. Considering variation and aging in a full chip design methodology at system level. In *Electronic System Level Synthesis Conference (ESLsyn), Proceedings of the 2014*, pages 1–6. IEEE, 2014.

[44] Paula Herber and Sabine Glesner. A HW/SW co-verification framework for SystemC. *ACM Transactions on Embedded Computing Systems (TECS)*, 12(1s):61, 2013.

[45] Wei Huang, Kevin Skadron, Sudhanva Gurumurthi, Robert J Ribando, and Mircea R Stan. Differentiating the roles of IR measurement and simulation for power and temperature-aware design. In *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, pages 1–10. IEEE, 2009.

[46] Andrew B Kahng, Bin Li, Li-Shiuan Peh, and Kambiz Samadi. ORION 2.0: a fast and accurate NoC power and area model for early-stage design space exploration. In *Proceedings of the conference on Design, Automation and Test in Europe*, pages 423–428. European Design and Automation Association, 2009.

[47] Andrew B Kahng, Bin Li, Li-Shiuan Peh, and Kambiz Samadi. ORION 2.0: a fast and accurate NoC power and area model for early-stage design space exploration. In *Proceedings of the conference on Design, Automation and Test in Europe*, pages 423–428. European Design and Automation Association, 2009.

[48] Michael Keating, David Flynn, Rob Aitken, Alan Gibbons, and Kaijian Shi. *Low power methodology manual: for System-on-Chip design*. Springer Publishing Company, Incorporated, 2007.

[49] Felipe Klein, Rodolfo Azevedo, Luiz Santos, and Guido Araujo. Systemc-based power evaluation with PowerSC. *Electronic System Level Design*, pages 129–144, 2011.

[50] Lars Kosmann, Daniel Lorenz, Axel Reimer, and Wolfgang Nebel. Enabling energy-aware design decisions for behavioural descriptions containing black-box ip-components.

In *Power and Timing Modeling, Optimization and Simulation (PATMOS), 2013 23rd International Workshop on*, pages 51–58. IEEE, 2013.

[51] Fadi J Kurdahi, Daniel D Gajski, Champaka Ramachandran, and Viraphol Chaiyakul. Linking register-transfer and physical levels of design. *IEICE Transactions on Information and Systems*, 76(9):991–1005, 1993.

[52] Benjamin C Lee and David M Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *ACM SIGPLAN Notices*, volume 41, pages 185–194. ACM, 2006.

[53] Ikhwan Lee, Hyunsuk Kim, Peng Yang, Sungjoo Yoo, Eui-Young Chung, Kyu-Myung Choi, Jeong-Taek Kong, and Soo-Kwan Eo. PowerViP: Soc power estimation framework at transaction level. In *Proceedings of the 2006 Asia and South Pacific Design Automation Conference*, pages 551–558. IEEE Press, 2006.

[54] Sheng Li, Jung Ho Ahn, Richard D Strong, Jay B Brockman, Dean M Tullsen, and Norman P Jouppi. McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pages 469–480. IEEE, 2009.

[55] Thorsten Grötkerand Stan Liao, Grant Martin, Stuart Swan, and Thorsten Grötker. *System design with SystemC*. Springer, 2002.

[56] Daniel Lorenz, Philipp A Hartmann, Kim Grüttner, and Wolfgang Nebel. Non-invasive power simulation at system-level with systemc. *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation*, pages 21–31, 2013.

[57] Mahesh Mamidipaka and Nikil Dutt. eCACTI: An enhanced power estimation model for on-chip caches. *Center for Embedded Computer Systems, Tech. Rep. TR04-28*, 2004.

[58] Milo MK Martin, Daniel J Sorin, Bradford M Beckmann, Michael R Marty, Min Xu, Alaa R Alameldeen, Kevin E Moore, Mark D Hill, and David A Wood. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. *ACM SIGARCH Computer Architecture News*, 33(4):92–99, 2005.

[59] Peter Marwedel. *Embedded systems design–embedded systems foundations of cyber-physical systems*. 2011.

[60] Jerónimo Castrillón Mazo and Rainer Leupers. *Programming Heterogeneous MPSoCs*. Springer, 2013.

[61] Deepak Mishra, Yasaman Samei, Nga Dang, R Domer, and Elaheh Bozorgzadeh. Multi-layer configuration exploration of MPSoCs for streaming applications. In *Electronic System Level Synthesis Conference (ESLsyn), 2012*, pages 38–43. IEEE, 2012.

[62] T Mudge, T Austin, and D Grunwald. SimPanalyzer: the simplescalar-arm power modeling project, 2004.

[63] Massoud Pedram. Power minimization in IC design: principles and applications. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 1(1):3–56, 1996.

[64] Michael D Powell, Arijit Biswas, Joel S Emer, Shubhendu S Mukherjee, Basit R Sheikh, and Shrirang Yardi. CAMP: A technique to estimate per-structure power at run-time using a few simple parameters. In *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, pages 289–300. IEEE, 2009.

[65] Jan Rabaey. *Low Power Design Essentials*. Springer, 2009.

[66] Anand Raghunathan, Niraj K Jha, and Sujit Dey. *High-level Power Analysis and Optimization*. Springer, 1998.

[67] Srivaths Ravi, Anand Raghunathan, and Srimat Chakradhar. Efficient RTL power

estimation for large designs. In *VLSI Design, 2003. Proceedings. 16th International Conference on*, pages 431–439. IEEE, 2003.

[68] Sherief Reda and Abdullah N Nowroz. Power modeling and characterization of computing devices: a survey. 2012.

[69] Santhosh-Kumar Rethinagiri, Oscar Palomar, Osman Unsal, Adrian Cristal, Rabie Ben-Atitallah, and Smail Niar. PETS: Power and energy estimation tool at system-level. In *Quality Electronic Design (ISQED), 2014 15th International Symposium on*, pages 535–542. IEEE, 2014.

[70] ITRS Roadmap. International Technology Roadmap for Semiconductors, 2009 edn. *Executive Summary. Semiconductor Industry Association*, 2009.

[71] Yasaman Samei and Rainer Doemer. MAVO: An Automated Framework for ESL Design Monitor, Analyze, Visualize and Optimize. 2014.

[72] Yasaman Samei and Rainer Dömer. Automated Estimation of Power Consumption for Rapid System Level Design. In *International Performance Computing and Communications Conference (IPCCC)*. IEEE, 2014.

[73] Yasaman Samei and Rainer Dömer. PowerMonitor: A Versatile API for Automated Power-Aware ESL Design. In *Forum on Specification & Design Languages (FDL)*. IEEE, 2014.

[74] Christoph Schumacher, Jan Henrik Weinstock, Rainer Leupers, and Gerd Ascheid. Scandal: Systemc analysis for nondeterminism anomalies. In *Forum on Specification and Design Languages (FDL)*, pages 112–119. IEEE, 2012.

[75] Alper Sen, Vinit Ogale, and Magdy S Abadir. Predictive runtime verification of multiprocessor SoCs in SystemC. In *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*, pages 948–953. IEEE, 2008.

[76] Premkishore Shivakumar and Norman P Jouppi. Cacti 3.0: An integrated cache timing, power, and area model. Technical report, Technical Report 2001/2, Compaq Computer Corporation, 2001.

[77] PowerMixer User Manual. http://www.tinnotek.com.tw.

[78] Vivek Tiwari, Sharad Malik, and Andrew Wolfe. Power analysis of embedded software: a first step towards software power minimization. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 2(4):437–445, 1994.

[79] Chiraz Trabelsi, Rabie Ben Atitallah, Samy Meftali, Jean-Luc Dekeyser, and Abderrazek Jemai. A model-driven approach for hybrid power estimation in embedded systems design. *EURASIP Journal on Embedded Systems*, 2011(1):569031, 2011.

[80] Claus Traulsen, Jérôme Cornet, Matthieu Moy, and Florence Maraninchi. A SystemC/TLM semantics in Promela and its possible applications. In *Model Checking Software*, pages 204–222. Springer, 2007.

[81] R Ubal, J Sahuquillo, S Petit, and P López. Multi2sim: A simulation framework to evaluate multicore-multithread processors. In *IEEE 19th International Symposium on Computer Architecture and High Performance computing, page (s)*, pages 62–68, 2007.

[82] GB Vece, Massimo Conti, and Simone Orcioni. PK tool 2.0: a SystemC environment for high level power estimation. In *Electronics, Circuits and Systems, 2005. ICECS 2005. 12th IEEE International Conference on*, pages 1–4. IEEE, 2005.

[83] Marian Verhelst and Wim Dehaene. *Energy Scalable Radio Design: For Pulsed UWB Communication and Ranging*. Springer, 2009.

[84] Ines Viskic and Rainer Domer. A flexible, syntax independent representation (SIR) for system level design models. In *Digital System Design: Architectures, Methods and Tools, 2006. DSD 2006. 9th EUROMICRO Conference on*, pages 288–294. IEEE, 2006.

[85] Steven JE Wilton and Norman P Jouppi. CACTI: An enhanced cache access and cycle time model. *Solid-State Circuits, IEEE Journal of*, 31(5):677–688, 1996.

[86] Wu Ye, Narayanan Vijaykrishnan, Mahmut Kandemir, and Mary Jane Irwin. The design and use of Simplepower: a cycle-accurate energy estimation tool. In *Proceedings of the 37th Annual Design Automation Conference*, pages 340–345. ACM, 2000.