The Dissertation Committee for Dongwook Lee
certifies that this is the approved version of the following dissertation:

Learning-Based System-Level Power Modeling of Hardware IPs

Committee:

Andreas Gerstlauer, Supervisor

Jacob A. Abraham

Lizy K. John

Keshav Pingali

Taemin Kim

Learning-Based System-Level Power Modeling of Hardware IPs

by

Dongwook Lee, B.S., M.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2017

To my beloved family.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my advisor, Prof. Andreas Gerstlauer. Without his profound insights, guidance, and support, this research and thesis could not have been possible. Through his many roles as an advisor, an engineer, and a researcher, he has had a significant influence on my professional development. I would also like to thank my committee members, Prof. Jacob Abraham, Prof. Lizy K. John, Prof. Keshav Pingali, and Dr. Taemin Kim, for their invaluable comments and suggestions. In addition, I wish to thank my friends, colleagues and collaborators.

Learning-Based System-Level Power Modeling of Hardware IPs

Publication No. _____

Dongwook Lee, Ph.D.
The University of Texas at Austin, 2017

Supervisor: Andreas Gerstlauer

Accurate power models for hardware components at high levels of abstraction are a critical component to enable system-level power analysis and optimization. Virtual platform prototypes are widely utilized to support early system-level design space exploration. There is, however, a lack of accurate and fast power models of hardware components at such high-levels of abstraction.

In this dissertation, we present novel learning-based approaches for extending fast functional simulation models of white-, gray-, and black-box custom hardware intellectual property components (IPs) with accurate power estimates. Depending on the observability, we extend high-level functional models with the capability to capture data-dependent resource, block, or I/O activity without a significant loss in simulation speed. We further leverage state-of-the-art machine learning techniques to synthesize abstract power models that can

predict cycle-, block-, and invocation-level power from low-level hardware implementations, where we introduce novel structural decomposition techniques to reduce model complexities and increase estimation accuracy.

Our white-box approach integrates with existing high-level synthesis (HLS) tools to automatically extract resource mapping information, which is used to trace data-dependent resource-level activity and drive a cycle-accurate online power-performance model during functional simulation. Our gray-box approach supports power estimation at coarser basic block granularity. It uses only limited information about block inputs and outputs to extract light-weight block-level activity from a functional simulation and drive a basic block-level power model that utilizes a control flow decomposition to improve accuracy and speed. It is faster than cycle-level models, while providing a finer granularity than invocation-level models, which allows to further navigate accuracy and speed trade-offs. We finally propose a novel approach for extending behavioral models of black-box hardware IPs with an invocation-level power estimate. Our black-box model only uses input and output history to track data-dependent pipeline behavior, where we introduce a specialized ensemble learning that is composed out of individually selected cycle-by-cycle models with reduced complexity and increased accuracy. The proposed approaches are fully automated by integrating with existing, commercial HLS tools for custom hardware synthesized by HLS. Results of applying our approaches to various industrial-strength design examples show that our power models can predict cycle-, basic block-, and invocation-level power consumption to within 10%,

9%, and 3% of a commercial gate-level power estimation tool, respectively, all while running at several order of magnitude faster speeds of 1-10Mcycles/sec.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The continued rise in hardware and software complexities of embedded on-chip systems has necessitated raising the design process to higher levels of abstraction. At the same time, energy efficiency has become a critical design concern. To address this challenge, heterogeneous multi-processor architectures utilizing massive custom hardware acceleration have recently emerged [1–3]. Depending on applications, custom hardware accelerators can take more than 25% of total area and power consumption of such accelerator-rich architectures [1]. Fast and accurate system-level power estimation approaches are needed to drive associated validation and optimization. Virtual platform models capable of simulating whole systems are widely employed to provide rapid feedback for design space exploration. Instead of slow co-simulation with low-level register-transfer level (RTL) or cycle-accurate models of custom hardware accelerators, intellectual property components (IPs) and processors, a purely functional modeling of hardware and software behavior is typically utilized.

To support efficient exploration, there is a need for extended models that can provide quick yet accurate estimates of critical system metrics such as

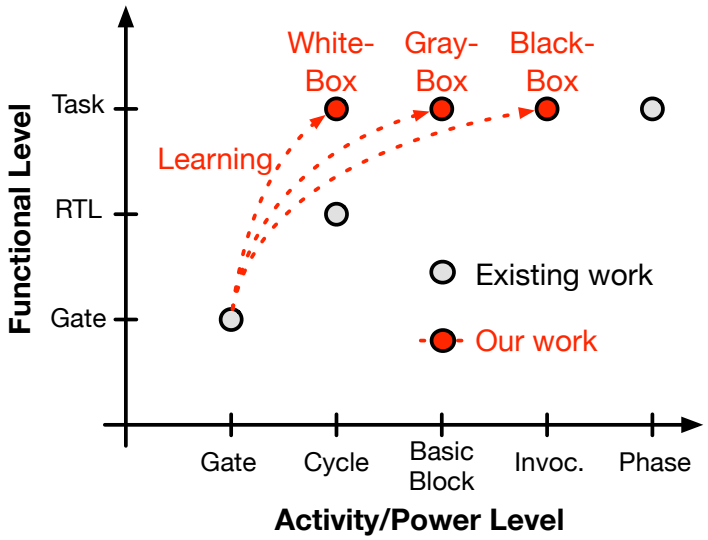


Figure 1.1: Power modeling space.

performance and power at a high level of abstraction. However, the modeling gap between fast, purely functional models for integration into virtual platforms and corresponding physical hardware implementations makes accurate power modeling challenging.

Figure 1.1 classifies power modeling approaches based on the granularity and abstraction level of their functional simulation versus activity and power estimation. At the lowest level, a detailed but expensive simulation of gate-level switching activity is used to estimate gate-level power consumption. Various power modeling approaches at higher levels of abstraction have been proposed. Most previous work at the system level utilizes a fast functional C/C++ task simulation to drive state-based power estimations that only model transitions between different coarse-grain operation modes [4–9].

Other approaches use accurate but slow activity estimation at a finer micro-architecture or RTL granularity. More recently, solutions at the intermediate representation (IR) level have emerged [10–14]. However, they similarly rely on slow, fine-grain simulation of the cycle-by-cycle behavior of individually scheduled IR operations in control/dataflow graph (CDFG) or finite state machine with data (FSMD) form to obtain accurate results.

Existing approaches all estimate power at the same level of detail at which the functionality of hardware is modeled. This allows a detailed functional simulation to drive an accurate, potentially data-dependent power estimation model, but also creates a fundamental trade-off between speed and accuracy depending on the simulation granularity.

The goal of this dissertation is to explore approaches that bridge the modeling gap by enabling fast and accurate power estimation at the system level. Instead of detailed micro-architecture or FSMD/CDFG simulation, we statically synthesize data-dependent switching activity-based power models of a given gate-level implementation using machine learning approaches. Based on given architectural information, traces of signal transitions captured from a functional simulation are then used to drive the abstracted power model.

1.1 Thesis Statement

In this dissertation, we demonstrate that it is possible to provide fine-grain, data-dependent power models for custom hardware components that can run at speeds close to a fast, purely functional simulation while being

able to achieve close to gate-level accuracy. To achieve this objective, we extract data-dependent activity features from functional hardware models based on given architecture information. Depending on the observability of hardware internals and their mapping to high-level constructs, extended white-, gray-, or black-box models are able to capture data-dependent operation, basic block, or I/O activity, respectively. Extracted activity data is then used to drive corresponding cycle-, block-, or invocation-level power models, where we statically synthesize data-dependent, activity-based power models at three different levels from a given gate-level implementation using machine learning approaches. This allows the traces of signal transitions captured from a functional simulation to be used to dynamically or statically drive an accurate, data-dependent power model at different prediction granularities.

1.2 Overview of Power Modeling Flow

Figure 1.2 shows an overview of our proposed power modeling flow. The inputs to the flow are a high-level functional simulation model of a hardware component, its corresponding gate-level implementation and optional micro-architecture mapping information. Depending on the observability, architecture information can consist of a complete mapping of high-level operations into RTL states and resources, the mapping of basic block inputs and outputs to resources and ports, or only limited information about the mapping of external I/O, e.g. in case of black-box IPs. Micro-architecture information can be manually provided or automatically extracted during synthesis. In our

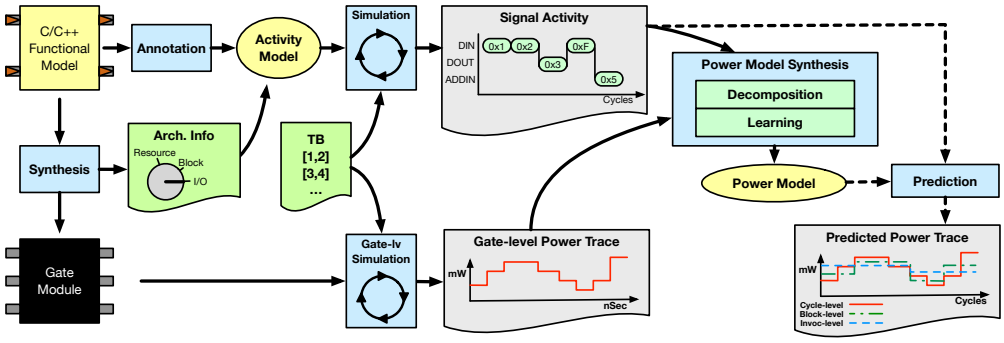


Figure 1.2: Power modeling flow.

flow, we integrate with existing, commercial high-level synthesis (HLS) tools to provide a fully automated power model generation for custom hardware synthesized by HLS.

Using micro-architecture mapping information, we annotate the high-level functional simulation code with the ability to capture activity traces of individual operand and result, basic block I/O or external I/O value transitions. In a training phase, the given gate-level model and the generated activity model are then simulated with the same input vectors. Power synthesis utilizes operation, block, or I/O activity traces from the high-level simulation together with cycle-level power traces from gate-level estimation to learn a power model. Instead of building a single power model, the synthesis flow decomposes power models into multiple models. Each decomposed power model is further simplified using a feature selection to reduce the amount of switching information that needs to be collected. In the process, the activity model is also simplified by removing unnecessary signal tracing not utilized after fea-

ture selection. In the prediction phase, the synthesized power models are then used to estimate data-dependent cycle-, block-, or invocation-level power traces from corresponding operation, basic block, or external I/O activity captured in high-level simulations.

1.2.1 Activity Model Generation

Our flow uses an annotation process to first refine a high-level C/C++ hardware functional model into an activity model. Depending on available architecture information, the refined activity model supports three different levels of switching activity tracing: individual resources, blocks, or only external I/Os. Figure 1.3 shows an overview of our activity model generation flow, accompanied by representative models and code snippets at various stages. In our framework, we synthesize a given functional hardware model down to an RTL description using a standard high-level synthesis process. In the process, we extract the IR of the design generated by the front-end compiler for high-level synthesis. Working at the IR level allows us to accurately reflect source-level optimizations, such as bit width reductions that affect tracking of internal signals in the synthesized RTL datapath. At the same time, the IR is extracted in C/C++ form before back-end synthesis in the HLS tool, i.e. it remains at a fast functional level. The IR code is further synthesized into an RTL implementation by the HLS tool. In this process, we automatically extract architecture information in the form of an extensible markup language (XML) file that stores mapping information between the IR and the

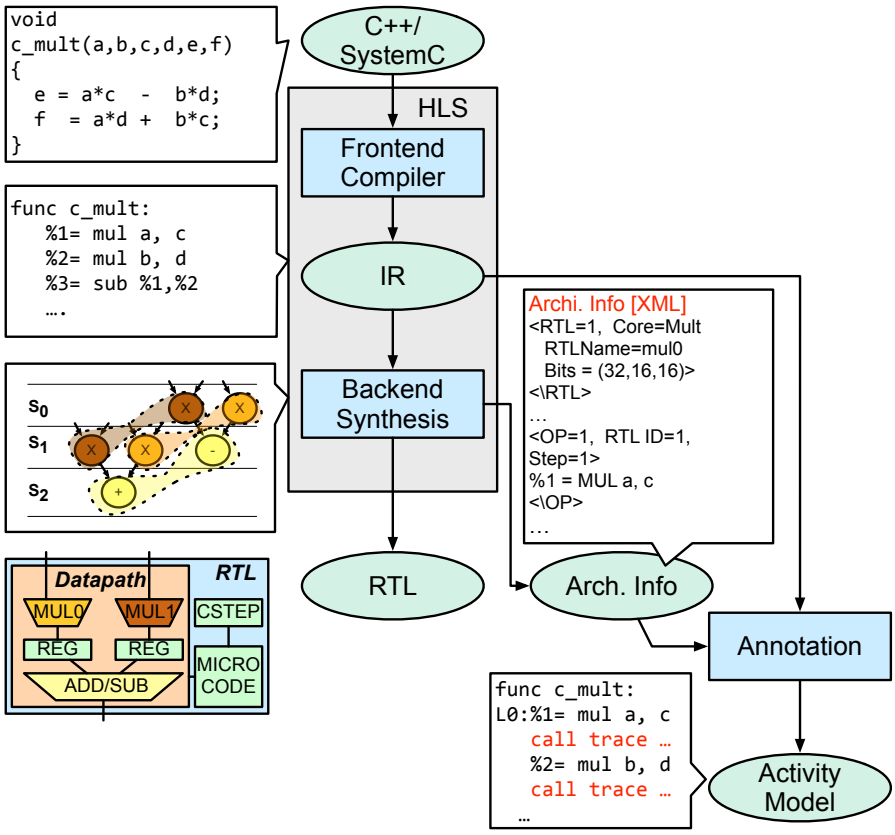


Figure 1.3: Activity model generation flow.

synthesized RTL implementation. Mapping information can be automatically generated during HLS as in our case or optionally manually provided. Depending on observability, it captures the mapping of IR operations to RTL control steps and datapath, the mapping of basic block inputs and outputs to resources and ports, or the mapping of functional interfaces to external I/O ports. The annotation process then automatically inserts corresponding signal *trace*() functions to generate an activity model that allows capturing cycle-by-cycle switching activity of individual datapath resources, basic block-by-basic

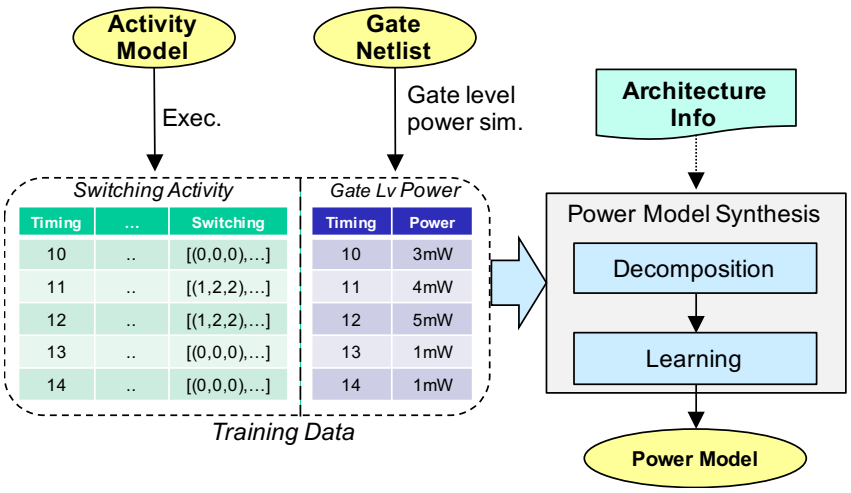


Figure 1.4: Power model synthesis flow.

block activity of block input and outputs, or invocation-by-invocation activity of external I/O during functional IR simulation. Depending on available architecture information, the refined activity model supports three different levels of switching activity tracing: individual resources, blocks, or only external I/Os. We will describe details of annotation and activity tracing for each model in Chapter 3, Chapter 4, and Chapter 5, respectively.

1.2.2 Power Model Synthesis Flow

After collecting switching activity traces from the activity model using simulations of a training set, power models are then synthesized in a one-time offline learning process. Figure 1.4 shows an overview of power model synthesis flow. A full power model is first decomposed into several simpler models using architecture information. Each decomposed power model is further simplified

using a feature selection to reduce the amount of switching information that needs to be collected. Power models are then trained from given power and activity traces. Activity traces are collected from activity model simulation and contain resource-, block-, or external I/O-level switching vectors. Power traces contain actual power measurements from an equivalent gate-level simulation.

We will describe the power model synthesis processes utilizing state-of-the-art machine learning techniques for cycle-, block-, and invocation-level power models corresponding to white-, gray-, and black-box hardware IPs in Chapter 3, Chapter 4, and Chapter 5, respectively.

1.3 Contributions

In this dissertation, we present a comprehensive and fully automated power modeling framework that provides fast yet accurate learning-based power estimation at three levels of abstraction. In the following, we summarize the contributions presented in the subsequent chapters.

1.3.1 Power Modeling of White-box IPs

We propose an approach that extends fast functional custom hardware models of white-box custom hardware IPs with the ability to produce detailed, cycle-level timing and power estimates. Our approach is based on back-annotating behavioral hardware descriptions with a dynamic power and performance model that allows capturing cycle-accurate and data-dependent activity without a significant loss in simulation speed. By integrating with ex-

isting HLS flows, back-annotation is fully automated for custom hardware synthesized by HLS. We further leverage state-of-the-art machine learning techniques to synthesize abstract power models, where we introduce a structural decomposition technique to reduce model complexities and increase estimation accuracy. The specific contributions are:

- We develop a light-weight approach for extracting cycle-accurate signal transition information from a high-level functional simulation without the need for full architecture simulation.
- We introduce a novel approach for decomposing learning-based power models using scheduling and binding information to reduce model complexity while improving estimation accuracy.

1.3.2 Power Modeling of Gray-box IPs

We further introduce an intermediate gray-box approach that supports power estimation at basic block-level granularity. It utilizes less total switching activity and fewer invocations of the power model than cycle-level models, while providing a finer granularity than invocation-level models, which allows to further navigate estimation accuracy and speed trade-offs. The specific contributions are:

- Using only limited mapping information about basic block inputs and outputs, we develop a light-weight approach for extracting block-level activity from a functional simulation.

- We propose a basic block-level power model that utilizes a novel decomposition using control flow information to reduce model complexity while improving estimation accuracy.

1.3.3 Power Modeling of Black-box IPs

We finally propose a novel approach for extending behavioral models of black-box custom hardware IPs with an accurate invocation-level power estimate. Our model utilizes only external I/O activity captured from a transaction-level simulation to track data-dependent pipeline behavior. The specific contributions are:

- We introduce an approach to extract fast, data-dependent invocation-level power models from gate-level power traces, where models are driven only by transaction-level I/O activity that does not require simulation overhead for cycle-level trace rearrangement or cycle-by-cycle activity computation.

- We develop a specialized ensemble learning approach in which invocation-level power models are decomposed into individual cycle-by-cycle models for efficient training and accurate prediction.

1.4 Methodology

In order to evaluate the accuracy of our power models, we measure and compare against cycle-by-cycle power traces obtained using a commercial

gate-level power estimation tool. A given gate netlist implementation and the synthesized functional model are simulated with the same input vectors. We then measure the gate-level cycle-by-cycle power traces from gate-level net signal transition traces. For all comparisons, we use cycle-by-cycle mean abolute error (MAE) of values $P_{estimated}$ predicted by each model compared to power measured from gate-level simulations, normalized against average power over the full simulation using the following equation:

$$MAE\,[\%] = \frac{\frac{1}{n}\sum_{i=1}^{n}|P_{estimated,i} - P_{measured,i}|}{\frac{1}{n}\sum_{i=1}^{n}P_{measured,i}} \times 100 \tag{1.1}$$

To evaluate block-by-block and invocation-by-invocation MAE, we convert the gate-level cycle-by-cycle trace by assigning the average power dissipation of each basic block and invocation period to corresponding blocks and invocations, respectively. We then measured the basic block-by-basic block or invocation-by-invocation MAE using equation (1.1).

To evaluate average errors, we compute a difference of the measured and estimated average power consumption over the whole simulation. The difference is normalized against measured average power over the full simulation using the following equation:

$$Average\ Error\,[\%] = |1 - \frac{\frac{1}{n}\sum_{i=1}^{n}P_{estimated,i}}{\frac{1}{n}\sum_{i=1}^{n}P_{measured,i}}| \times 100 \tag{1.2}$$

To evaluate the simulation speed of our extended functional models, we measure the number of simulated cycles from gate netlist simulation. Based

on the total simulation runtime of our functional models, we present speed numbers as simulation throughput measured in cycles per second (cycles/sec) using the following equation:

$$Speed\,[cycles/sec] = \frac{Simulated\,Cycles}{Simulation\,Time} \tag{1.3}$$

1.5 Thesis Outline

The remainder of this dissertation is organized as follows. Chapter 2 reviews relevant prior work. Next, Chapter 3 presents a power model for white-box IPs, where we extend a high-level functional model with the capability to produce cycle-level power estimates using detailed architecture information. Chapter 4 introduces an intermediate gray-box approach that supports power estimation at basic block-level granularity using limited architecture information. Chapter 5 presents our approach for black-box IPs, which enables data-dependent power modeling without internal architecture information. Chapter 6 then summarizes and compares accuracy and speed of proposed white-, gray-, and black-box models. Finally, Chapter 7 concludes this dissertation and proposes directions of future research.

Chapter 2

Related Work

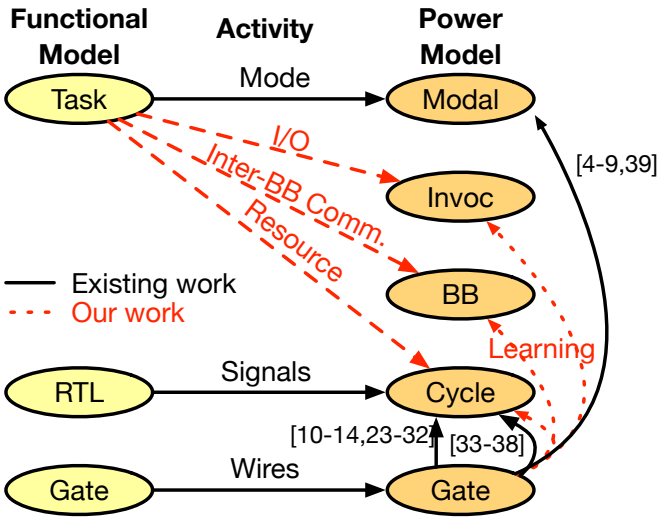


Figure 2.1: Power modeling approaches.

In this chapter, we briefly review prior power modeling work. Figure 2.1 shows a more detailed taxonomy and overview of existing power modeling work in relation to our approach. Traditional accurate power models are constructed by coupling gate-level simulations with gate library power models. To generate higher-level timing and energy models of custom hardware accelerators and processors, library or learning-based approaches can be utilized. In a library-based approach, an overall model is assembled from pre-characterized

component data [7-11, 19-28]. This enables rapid exploration but does not accurately account for all glue logic and implementation-level optimizations in a combined architecture.

In learning-based approaches, a RTL or detailed micro-architecture implementation is simulated in a sampling fashion to derive a regression-based model for a complete processor or each macro-block [29-34]. Such approaches can accurately reflect the behavior of the final implementation, but still require simulation at the RTL or micro-architecture level to extract internal signal information driving the generated models. By contrast, approaches that drive a learned power model from high-level functional task simulations are fast, but only allow to capture coarse-grained power transitions between phases [1-6, 35].

In all cases, a slow low-level detailed functional simulation allows to drive an accurate, data-dependent power estimation model while a fast high-level functional simulation only captures coarse-grained power transitions. This creates a fundamental trade-off between speed and accuracy depending on the granularity and level of the power model. We aim to drive fine-grained, data-dependent power models directly from high-level C/C++ functional simulations. Our approach supports both library-based and learning-based methods, where our focus is on the learning-based generation of light-weight implementation-level representations of complete hardware processors. We propose cycle-, basic block-, and invocation-level power models combined with extraction of resource, inter-basic block communication, and external I/O

activity from high-level functional simulations.

In the following, we will review previous power modeling works including low gate-level, RTL and micro-architecture power models, as well as high-level functional models, including a discussion of their performance and accuracy trade-offs. In addition, we discuss source-level software simulation approaches that provided some of the initial ideas for our proposed work.

2.1 Gate-Level Models

Instead of prohibitive slow circuit-level or SPICE simulations, gate-level power estimation is widely used for accurate power analysis. In such approaches, toggling activity of individual bit-level signals is collected during gate-level netlist simulation to drive a gate-level library power model [15–17]. This enables accurate, data-dependent, and fine-grain power estimation but requires detailed interconnect and logic timing computation to extract accurate bit-level signal transitions, which is too slow. To simplify timing and switching computation for each gate learning-based stochastic models for groups of combinational logic have been proposed [18]. Such gate-level power estimation is much faster than circuit-level approaches [19–21], but still too slow to estimate power consumption of complete, large-scale hardware implementation with many test vectors. To reduce gate-level simulation time, a fast average power estimation approach was recently proposed [22]. This approach samples snapshots of internal signals using an FPGA-accelerated RTL simulator and then drives gate-level power estimation tools with the small sampled snap-

shots. This reduces the overall gate-level simulation times, but introduces several hours of a FPGA compilation overhead.

2.2 RTL and Micro-Architecture Models

Many approaches has been proposed to collect resource- or block-level activity from a finer-grain micro-architecture or register-transfer level simulation. To generate corresponding power models of custom hardware accelerators and processors, library or learning-based approaches can then be utilized.

In library-based approaches, the activity traces of each fine-grain arithmetic and logic resource are collected from RTL [13, 23–25], CDFG-level [10–12, 14], or fine-grain micro-architecture [26] simulations and then drive corresponding resource-level power models. The RTL power estimation then extracts various characterizations (i.e. input and output transition probabilities or resource-level activity traces) from those traces and computes component-level power consumptions using either simple table-lookup based [27, 28], analytical [29, 30], or regression-based [31, 32] models.

In learning-based approaches, regression-based models for complete processors or macro blocks are pre-characterized using sampled gate-level power traces. Such regression-based models utilize the internal signals obtained from the RTL of complete processors or hardware accelerator IPs. Such approaches can accurately reflect the behavior of the final implementation using collected internal activity traces.

A key concern of regression-based approaches is managing model complexities without sacrificing accuracy. Existing approaches rely on sampling a subset of key signals or state variables that are identified either manually or in a trial-and-error process. PowerDepot [33] and PrEsto [34] build hardware IP power models using manually selected relevant signals, where PrEsto utilizes additional linear regression-based importance sampling. In [35,36], the important signals in the micro-processors are selected based on a learning process or singular value decomposition. Other approaches decompose the full power model into several parts based on manual decisions [37,38]. This requires detailed architectural knowledge or designer insight, which is often not available, especially for black-box IPs.

Both library and learning-based approaches enables cycle-level, data-dependent power estimations. However, all approaches require slow, fine-grain simulation of the cycle-by-cycle behavior of each resource or block, which is typically too slow to be integrated into virtual platforms at the system level.

2.3 System-Level Models

System-level component models are often only functionally equivalent ones, where necessary internal architectural information for fine-grain modeling is not available, especially in case of pre-designed IPs. The limited observability of such high-level, black-box models restricts power estimation to a coarse-grain state-based approach, where the projection of either given, documented states or state information estimated from external transaction events

only supports capturing coarse-grained power phase transitions between different operating modes, such as read and write modes in memories or buses. Many early work annotates a single average power consumption to each coarse-grained state using an automated characterization flow [4, 5]. Improving the accuracy of power models attached to high-level functional simulations that do not provide internal architecture information has been the focus of many researchers. Copty *et al.* [7] proposed characterizing pairs of coarse-grained states using statistical methods. Schürmans *et al.* [6, 9] extract the power state machines of processors and communication architectures using multiple observed or estimated architecture states and measured power traces. Kosmann *et al.* [39] generate power state machines by only observing state trace of external ports. To take into account data-dependent effects in power estimation of system-level black-box components, a corresponding extension of coarse-grain state-based models was recently proposed [8]. In this approach, cycle-level input switching activity information is utilized to refine states in which significant data-dependent power variations are observed. This requires augmenting state-based models with the ability to capture cycle-by-cycle activity, which introduces a significant overhead in the simulation. Furthermore, a simple linear regression is inherently limited in accuracy.

2.4 Source-Level Simulation

For software running on processors, so-called source-level or host-compiled modeling approaches have recently emerged as an alternative to instruction-set

or micro-architecture simulation. In such approaches, a source or IR model of the application is statically back-annotated with timing and energy estimates extracted from low-level simulations [40]. In [41–43], a constant or statistical energy consumption at the granularity of instructions, source-level operations, program phases or processor states are annotated to sources. The authors in [43] shows that such static back-annotation approach can achieve 400x speedup compared to a target ISS with up to 13% error compared to reference simulators. More recent work [40] proposed characterizing blocks in static pairs with low-level reference micro-architecture power models, which enables low-level accurate power estimation at fast source-level simulation. However, such static back-annotation approaches are typically performed at the basic block level, which is only able to capture control-dependent power behavior. Our proposed approach is motivated by host-compiled software models, but also aimed at accurately capturing data-dependent power effects. Instead of back-annotating static per block estimates, we annotate the functional simulation with dynamic, data-dependent cycle-, block- or invocation-level power models.

Chapter 3

Power Modeling of White Box IPs

Accurately capturing cycle-level power variations is important for many design decisions. Previous approaches for data-dependent cycle-level power models require a tight coupling with cycle-level functional models such as fine-grain micro-architecture simulators or RTL implementations. By contrast, we propose an approach that extends fast functional hardware models with the ability to produce detailed, cycle-level timing and power estimates.

In this chapter, we introduce a framework that realizes such a novel, fast yet accurate cycle-level power modeling for white-box hardware IPs [44]. We propose a light-weight approach for extracting white-box, resource-level signal activity tracing from a high-level functional simulation without the need for full architecture simulation. We leverage machine learning technique to synthesize cycle-level power model. We further propose a novel approach for decomposing power models using scheduling and binding information to reduce model complexity while improving estimation accuracy.

The rest of the chapter is organized as follows: we describe the details of our resource-level activity model generation and cycle-level power model synthesis in Section 3.1 and Section 3.2. Next, in Section 3.3, we evaluate

accuracy and speed of the power models with a set of industrial-strength design examples. Finally, Section 3.4 concludes the chapter with a summary.

3.1 Resource-Level Activity Model Generation

In the proposed power modeling flow (Figure 1.2), a given behavioral hardware model is first synthesized down to an RTL description using a standard high-level synthesis process. In the process, FSMD-level micro-architecture information is automatically extracted and the annotation process then refines a high-level C/C++ hardware functional model into an activity model (Figure 1.3). In the following, we describe the resource-level annotation process and corresponding activity computation.

3.1.1 Annotation for White-Box IPs

In a white-box case, we support capturing cycle-accurate activity of RTL datapath resources, such as adders and multipliers, during high-level functional simulation by back-annotating abstract micro-architecture information into the IR. We assume that micro-architecture mapping information is provided, where we can extract an FSMD-level description from the HLS tool. The extracted architecture information includes each IR operation node's resource scheduling, binding, and bit width information. Based on this information, the annotation process inserts *trace*() functions that store the operands and results of each IR operation together with the scheduled control state and bound resource ID to compute switching activity. We capture the flow of data

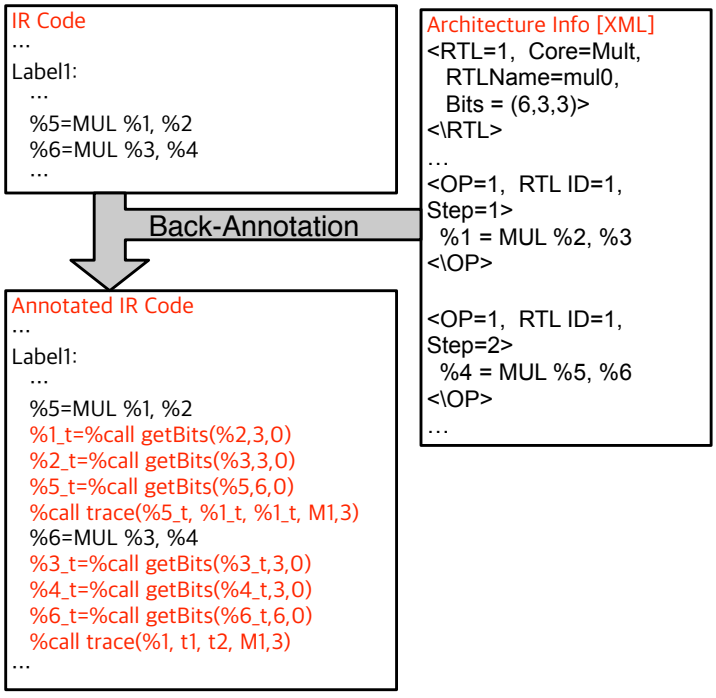


Figure 3.1: Resource-level annotation process.

and associated switching activity by tracing IR operands and results. To map data activity into signal transitions of actual hardware resources, we include resource scheduling and binding information in the captured traces. In addition, bit width information is annotated to extract the actual number of bits utilized in hardware. This information is then used to track cycle-by-cycle activity of each resource while taking into account resource sharing and other back-end synthesis optimizations.

Figure 3.1 shows code snippets for the signal extraction and tracing process. The mapping information is provided by the HLS tool in the form of an FSMD architecture XML file. The mapping information file stores each

RTL resource's bit width information and resource ID and each operation node's scheduling and binding. In back-annotation process, we annotate the IR code with calls to a *trace*() function, which stores the operands and results of each IR operation together with the scheduled control state and bound resource ID. To take bit width optimizations into account, an additional *getBits*() function is annotated to extract the actual number of bits utilized in the hardware.

3.1.2 Resource-Level Activity Computation

The hardware implementation generally exploits operation-level parallelism and scheduling flexibility to maximize performance under given resource or timing constraints. As a result, operators in the IR are not necessarily simulated in the same order in which they execute in the final hardware. Figure 3.2 and Figure 3.3 show such intra- and inter-block level out of order execution scenarios, respectively.

In order to rearrange out-of-order execution traces captured in the IR simulation into in-order traces for hardware estimation, we perform an online reordering of traced information using annotated scheduling and binding information. As shown in Figure 3.2, the execution order of two operators in the same basic block can be reversed in the hardware implementation if there is no dependency between the operations. To rearrange the trace, we utilize a global signal table and a trace reordering buffer. The global table tracks signal values of all hardware resources in the most recent cycle. The

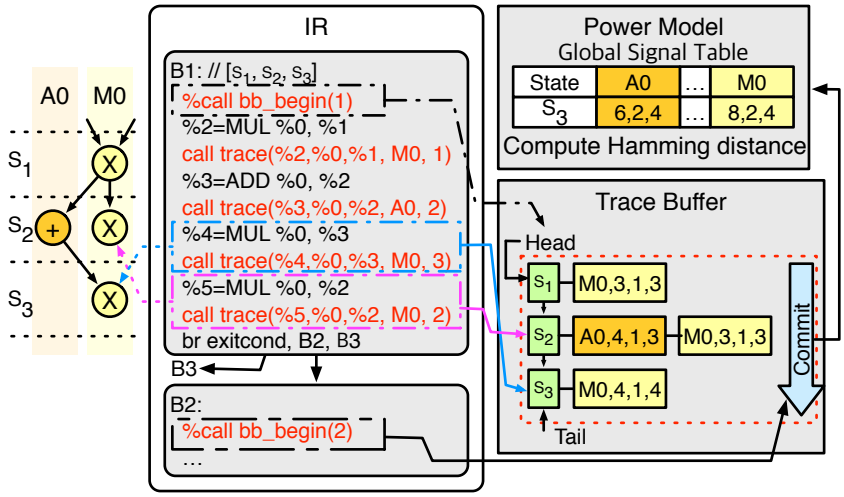


Figure 3.2: Intra-block level out of order execution scenarios.

trace buffer temporally stores and reorders signal updates associated with the current basic block. It consists of control state tags and corresponding signal trace lists. Each entry in the signal trace lists contains the utilized resource ID together with operands and result of the operation. At the beginning of each basic block, an additional function is annotated to initialize the buffer and insert state tags corresponding to the block's control states. Within the block, each call to the *trace*() function then attaches a new entry to the signal trace linked list corresponding to the annotated control state. At the end of the current and beginning of the next basic block, all signals updated in each control steps are sequentially committed to the global signal table, the head of the buffer is moved to the tail, all current control step and trace lists are discarded, and new control state tags assigned to the next block are inserted. In this process, the Hamming distances of all signals toggling in each control

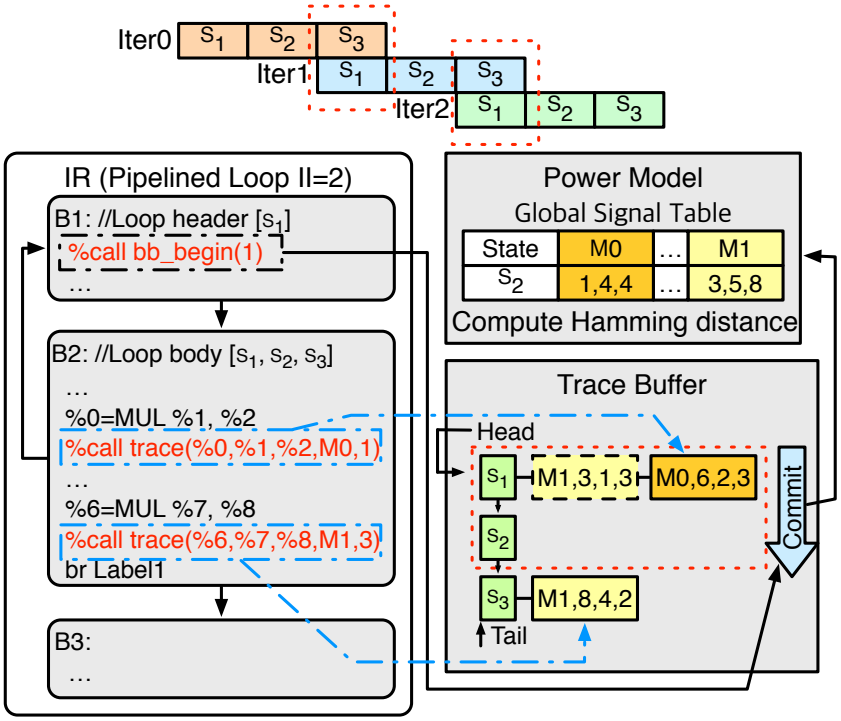


Figure 3.3: Inter-block level out of order execution scenarios.

step are computed, and this switching activity information is committed to either a tracing file or the final power model. In addition, for performance estimation, a global cycle counter is increased by the number of cycles spent on the block.

As shown in Figure 3.3, the execution of basic blocks can be overlapped in the case of pipelined hardware loops. This results in some operators in the second iteration to be executed before the last operator in the first iteration. In [44], we had introduced an additional intermediate pipeline buffer that retains signal traces of previous iterations to emulate the pipeline structure.

To improve simulation speed, we now account for such pipeline effects by instead controlling the head and tail management in the trace buffer itself. When first entering the header block of a pipelined loop, control state tags for a single iteration of the loop body block are inserted into the buffer. If a pipelined execution is detected, the trace buffer is not committed during execution of the loop body. Instead, during execution of the header block at the start of each new loop iteration, only the completed control steps, i.e. entries corresponding to the loop initiation interval (II) are committed, and the head is moved and entries are discarded accordingly. Remaining entries are retained and their state tags relabeled to overlap with the start of the next iteration. Finally, new control state tags for the bottom part of the loop body are inserted into the trace buffer. With each such iteration, new traces will be added to the remaining buffer contents, which will contain uncommitted signal data from previous iterations. After the end of execution of a loop, all remaining entries in the buffer are committed. Loop information (II as well as IDs of all loop header and body blocks) is automatically extracted from the HLS tool together with other scheduling and binding information. Overall, this approach allows us to accurately trace the signal transitions of hardware resources without the need for a slow lockstep pipeline simulation.

3.2 Cycle-Level Power Model Synthesis

After collecting switching activity traces from the activity model using simulations of a training set, power models are then synthesized in a one-

time offline learning process. In the following, we describe the cycle-level power model associated with resource activity including proposed power model synthesis processes utilizing state-of-the-art machine learning techniques.

3.2.1 Cycle-Level Power Modeling and Decomposition

Previous approaches for power estimation at the gate, RTL or micro-architecture level mostly choose a linear function to model the relation between the internal signal switching activity and power consumption of a hardware component. Given the internal and external signal switching activity column vector $\mathbf{a}(t)$ at time t, power consumption $p(t)$ can be modeled as

$$p(t) = \boldsymbol{\theta} \cdot \mathbf{a}(t), \tag{3.1}$$

where $\boldsymbol{\theta}$ denotes a coefficient row vector. To simplify the model, we assume that related pins, e.g. of buses are grouped, and Hamming distances within a group are utilized as an alternative to individual bit-wise switching activity. With this assumption, power behavior of complex arithmetic units is generally not linear [25], but without loss of generality, we use a linear model for the following model derivations.

Ignoring glitching or asynchronous activities, we can convert the continuous power function into a discrete cycle-level model. In general, average power consumption p_n in cycle n can be modeled as

$$p_n = \frac{1}{T} \int_{(n-1)T}^{nT} p(t)\, dt = \boldsymbol{\theta} \cdot \mathbf{a}(nT) = \boldsymbol{\theta} \cdot \mathbf{a}_n = P_{\mathrm{CS}}(\mathbf{a}_n), \tag{3.2}$$

where $\mathbf{a}_n$ is a discrete activity vector. We utilize resource-level activity vectors captured during functional tracing to drive a single cycle-level power model $P_{\mathrm{CS}}(\mathbf{a}_n)$.

The complexity of the power model in (3.2) is directly proportional to the dimension of the activity vector, where high dimensionality may create generalization errors in learning processes. To avoid over-fitting, feature sampling, which reduces model dimensions by selecting a key subset of signals, can be utilized, but this may result in a loss of accuracy. As an alternative to traditional feature selection, we introduce a structural model decomposition that uses architectural information to reduce unnecessary signals while improving accuracy.

In white-box models, hardware can be described in FSMD form. Given a finite set of FSMD states S, where the state executed in cycle n is defined as s_n, the power consumption in a given cycle n is dependent on resource utilization in FSMD state s_n. Further, given a finite set of hardware resources R, a resource scheduling and binding function can be defined as $m : S \times R \rightarrow \{0, 1\}$. For instance, $m(r, s) = 1$ indicates that resource r is utilized in the state s. With such mapping information, we can formulate the power consumption in a given cycle n in the following manner:

$$p_n = \sum_{r \in R} m(r, s_n)\, \boldsymbol{\theta}_r \cdot \mathbf{a}_{n,r} = \sum_{r \in R} \boldsymbol{\theta}'_{s_n,r} \cdot \mathbf{a}_{n,r}, \tag{3.3}$$

where $\boldsymbol{\theta}_r$ and $\mathbf{a}_{n,r}$ denote the coefficient and switching activity subvectors corresponding to resource r, respectively. In this formulation, the coefficient

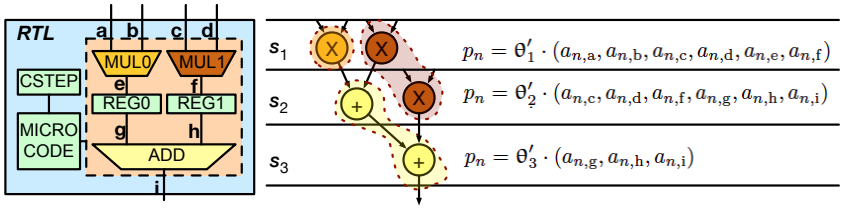


Figure 3.4: Example of power model decomposition.

factors vectors $\boldsymbol{\theta}_{s_n,r} = m(r, s_n)\ \boldsymbol{\theta}'_{s_n,r}$ are not only resource, but also state-dependent. Coefficients are masked and zeroed depending on resources utilized in each state. Crucially, such a re-formulation also allows unmasked entries to vary in order to be able to account for any power consumption of resources as well as connected control and glue logic being dependent on the control state.

With this, we can decompose equation (3.3) into separate and independent, decomposed cycle-level power models for each control state P_{CD,s_n}. In the process, we can further exploit mapping information $m(r, s)$ to identify and remove unnecessary signals $\mathbf{a}_{n,r}$ corresponding to unused resources r and thus masked activity in a particular state s_n:

$$p_n = P_{\mathrm{CD},s_n}(\mathbf{a'}_{s_n,n}) = \boldsymbol{\theta}'_{s_n} \cdot \mathbf{a}'_{s_n,n}, \tag{3.4}$$

where $\mathbf{a}'_{s_n,n} = (\mathbf{a}_{n,r} | r : m(r, s_n) \neq 0)$ is a subvector composed of activity of the signals used in the state s_n. We illustrate this with the help of a small example. Figure 3.4 shows a hardware micro-architecture in which three resources are allocated (*MUL0*, *MUL1* and *ADD*). The power consumption of the complete hardware processor can be estimated using a single cycle-level model (P_{CS})

from (3.2) using all switching vectors connecting to all resources. By contrast, the decomposed power model (P_{CD}) of a given control state instead utilizes the much smaller subset of signals connecting the resources scheduled in the given state only. For example, the power consumption of state S_3 can be estimated with three signals instead of all nine switching vectors. As such, a power model decomposition based on structural micro-architecture information is able to reduce the complexity of the model with little to no information loss. At the same time, it also allows for state-dependent variations in coefficients $\boldsymbol{\theta}'_s$ that can account for differences in power consumption of resources and other shared logic.

3.2.2 Feature Selection

Decomposition based on the FSMD information still has limitations in handling states with high resource utilization, such as pipelined states with many scheduled operators. Moreover, decomposition still requires all signals to be traced across states, which decreases simulation speed. We therefore apply additional feature selection to further reduce complexity and improve estimation latency. As part of basic timing back-annotation, we already select only key signals to trace based on the expected power contribution of resources in the micro-architecture. The power consumption of complex units, such as adders, multipliers or registers will be much higher with larger variations than the power of simple logic units, such as multiplexers or bitwise logic operators. Hence, we only sample the signals connected to such resources. Based on the

resource mapping information extracted from the FSMD, we trace input and output signals for IR operations mapped to arithmetic units. To also take registers into account, we extract the variable mapping information from the FSMD and trace any outputs of operations that store their results in registers.

To further reduce feature sets, we additionally leverage a decision tree approach from machine learning [45]. Decision trees are well known for their ability to automatically determine relative importance of features from the training data. We apply such feature selection after model decomposition. Feature selection first trains a decision tree model, extracts the importance of the signals, and then selects the key signals that exceed a given threshold.

3.2.3 Learning

Each power model is trained from given power and activity traces using established machine learning algorithms. Activity traces contain cycle times, states and corresponding switching vectors. Power traces contain actual power measurements from an equivalent gate-level simulation for the same set of training inputs. Activity and power traces are partitioned into states and inputs based on decomposed power models in each control step. Each power model is then trained with the corresponding partitioned traces and checked for accuracy using cross-validation methods.

Power behavior of complex arithmetic units is generally not linear [25]. We thus support linear as well as non-linear regression models. Depending on hardware functionality, input data statistics and complexity of models,

a non-linear machine learning model can represent the power consumption behavior better than a typical linear least squares model. Our learning flow also supports a cross-validation based model selection to find the best accuracy power model for given a training set. In doing so, power model synthesis trains each available learning model with the given training vectors and picks the final model according to cross-validation scores.

For online power estimation, a regression model library is inserted into the activity model. At the start of hardware simulation, pre-compiled power model parameters, coefficients, and data structures are loaded into regression models. As part of this process, unnecessary signal tracing calls inserted during annotation process are removed to improve estimation simulation speed. At run-time, the power model then estimates the power consumption of the hardware implementation from the dynamically computed switching activities.

3.3 Experimental Results

We have implemented a fully automated realization of our power modeling flow. We integrated our flow with the Vivado HLS engine [46] utilizing the LLVM compiler framework [47] for automatic activity annotation, prediction insertion and IP model generation. Power model synthesis utilizes the scikit-learn [48] machine learning library for Python. For fast online prediction, we natively implemented C++ based power estimation models to reduce Python binding overhead.

We applied our flow to generate models for pipelined and non-pipelined

hardware designs of a 6x6 general matrix multiplication (GEMM), a 2D discrete cosine transform (DCT), a JPEG quantizer (Quant) and a weight computation block of a high dynamic range (HDR) imaging application [49]. The quantizer has two control inputs for choosing a quantization table and the image scaling quality. All hardware designs were synthesized using Synopsys Design Compiler [50] with the Nangate 45nm Open Cell Library [51] at 200Mhz clock frequency. Gate-level power was estimated using Synopsys PrimeTime PX [52] with VCD files generated from full gate-level simulation. We measured power consumption of logic gates and registers, but power consumption of memories is excluded. All experiments were performed on a quad-core Intel i7 workstation running at 3.5 GHz. To learn each power model, we used training sets generated from different random seeds or images. In all cases, training vectors were selected to guarantee 100% of lines of code coverage. To generate test vectors, the GEMM design was simulated with 5000 random test matrices. A 640x320, a 512x512, and a 200x100 24-bit RGB image are used to generate DCT, QUANT, and HDR test vectors, respectively. Three different quality factors and two different table setting are utilized to generate the test set for the QUANT design. Table 3.1 summarizes benchmarks and synthesis results including number of states in each design, execution cycles per invocation, key IR operators and (shared) RTL resources selected for annotation and tracing. To generate test vectors, GEMM design was simulated with 5000 random test matrices. A 640x320, a 512x512, and a 200x100 24-bit RGB image are used to generate DCT, QUANT, and HDR test vectors, respectively. Three different

Table 3.1: Benchmark summary for white-box IPs

	Pipe	States	Cycles per Invocation	Gates	RTL Resources	Traced IR Op.
GEMM	No	6	734	703	11	11
	Yes	4	436	964	20	20
DCT	No	23	179	7,007	88	139
	Yes	12	94	6,309	61	127
HDR	No	18	995	4,883	35	70
	Yes	20	825	7,887	41	104
QUANT	No	6	194	1,032	7	7
	Yes	4	68	1,035	8	8

Table 3.2: Train and test summary for white-box IPs

	Pipe	Train Invoc.	Test Invoc.	Total Test Cycles	Avg. Power
GEMM	No	2,000	5,000	3,670,000	0.36mW
	Yes	2,000	5,000	2,180,000	0.72mW
DCT	No	3,000	10,800	1,933,200	0.67mW
	Yes	3,000	10,800	1,015,200	2.05mW
HDR	No	988	1,200	1,194,000	0.81mW
	Yes	988	1,200	990,000	1.07mW
QUANT	No	3,600	12,288	7,150,452	0.24mW
	Yes	3,600	12,288	2,506,752	0.43mW

quality factors and two different table setting are utilized to generate the test set for the QUANT design. Table 3.2 summarizes the size of training and test sets, and the average power consumption of each test set simulation.

Figure 3.5 and Figure 3.6 show accuracy and speed of proposed decomposed cycle-level power models (CD) as compared to a single cycle-level power model (CS) across various benchmarks. We measured data-dependent cycle-by-cycle MAE of values predicted by each model compared to gate-level simu-

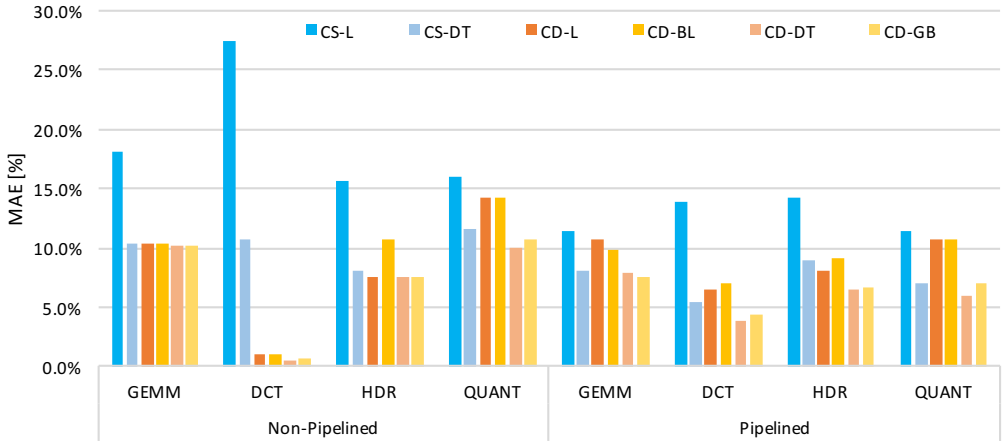


Figure 3.5: Cycle-by-cycle power accuracy.

lations, normalized against average power over the full simulation.We compare both power models utilizing either a least squares linear (CS-L and CD-L) or a decision tree (CS-DT and CD-DT) regression against a decomposed model using a linear Bayes ridged regression (CD-BL), or a gradient boosting regression composed of multiple decision trees (CD-GB). In all cases, we applied a decision tree based feature selection to remove uncorrelated features and then unused signals.

We can observe that, in all cases, linear decomposed models (CD-L) show on average 1.8x better accuracy than single cycle-level models with least squares regression (CS-L). The proposed structural decomposition technique results in up to 26% less MAE. A decision tree regression can improve the accuracy of the single model (CS-DT). However, it still shows higher errors in several cases, which indicates that decomposition is a key factor in improving model accuracy. Significant accuracy improvements are observed

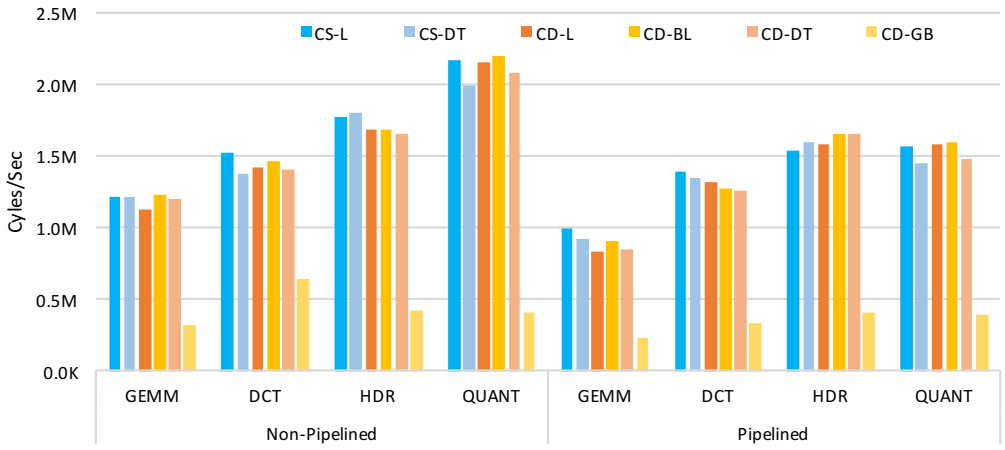


Figure 3.6: Estimation speed of models.

in the non-pipelined DCT case. In the non-pipelined DCT, there is a substantial power variation across states. It is generally hard to capture such state-dependent trends in a single cycle model. Compared to simpler designs (QUANT, GEMM), higher accuracy improvements are observed in complex hardware implementations (HDR, DCT), which again indicates that decomposition is more effective in large designs. Among all models, decomposed power models utilizing decision tree (CD-DT) or gradient boosting (CD-GB) regression show better accuracy than others. Linear models (CD-BL, CD-L) show the worst results in all cases, with up to 4.8% higher errors, where Bayesian models (CD-BL) generally perform similar or worse than standard least squares regressors.

Speed (Figure 3.6) generally depends on the complexity versus execution cycles of the design. Single models are slightly faster than decomposed ones on average. In the single models, more activity features are treated as

correlated and thus removed during feature selection, which results in significantly less accuracy but better speed. Models using gradient boosting regression (CD-GB) are on average 3.7x slower than others. Gradient boosting needs to call multiple subcomponent models, which generally introduces much larger prediction overhead. The decision tree model (CD-DT) is thereby 3.6x faster than a gradient boosting (CD-GB) one at similar accuracy. Least squares models (CD–L) are on average slightly faster, but decision tree models (CD-DT) provide on average 1.3x better accuracy. Overall, when comparing different regression methods and models, results show that a decomposed power model utilizing a decision tree regression (CD-DT) provides the best trade-off between accuracy and speed. The CD-DT model achieves on average 1.5Mcycles/sec at 93.4% accuracy. For further analysis, we utilize CS-DT and CD-DT models.

Table 3.3 and Table 3.4 further summarize and detail accuracy and speed of models across benchmarks. We measure cycle-by-cycle MAE, invocation-by-invocation MAE, and total average error across a full simulation. Overall, the CD-DT models improve accuracy over the CS-DT models by a factor of 1.4x on average across all error metrics. The CD-DT models estimate cycle-level and invocation-level power consumption within 10.1% and 3.6% compared to gate-level power results. In all cases, average errors across the whole simulation are below 1%.

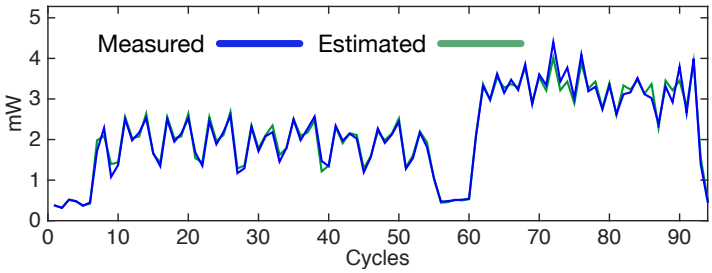
Table 3.3: Accuracy of cycle-level modeling

	Pipe	Cycle-by-Cycle MAE [%]		Invocation-by-Invocation MAE [%]		Average MAE [%]	
		CS-DT	CD-DT	CS-DT	CD-DT	CS-DT	CD-DT
GEMM	No	10.4%	10.1%	3.2%	3.1%	0.4%	0.4%
	Yes	8.0%	7.9%	2.3%	2.2%	0.1%	0.1%
DCT	No	10.7%	0.6%	0.8%	0.0%	0.5%	0.0%
	Yes	5.5%	3.9%	1.0%	1.1%	0.4%	0.5%
HDR	No	8.0%	7.6%	2.5%	2.0%	1.2%	0.9%
	Yes	9.0%	6.6%	2.6%	2.4%	1.3%	1.0%
QUANT	No	11.5%	10.0%	4.3%	3.6%	1.2%	0.1%
	Yes	7.0%	6.0%	1.8%	1.7%	0.0%	0.4%
Avg.	-	8.8%	6.6%	2.3%	2.0%	0.6%	0.4%

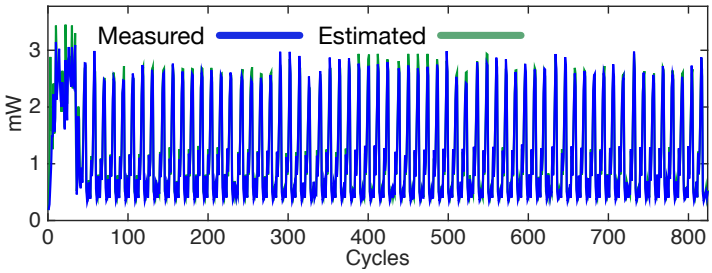
Table 3.4: Simulation speed of models [cycles/sec]

	Pipe	CS-DT	CD-DT	C Code	RTL	Gate
GEMM	No	1.21M	1.20M	220M	51K	0.61K
	Yes	0.92M	0.84M	130M	35K	0.36K
DCT	No	1.38M	1.40M	32M	16K	0.41K
	Yes	1.35M	1.25M	17M	5.9K	0.19K
HDR	No	1.80M	1.66M	32M	13K	0.28K
	Yes	1.60M	1.65M	27M	11K	0.20K
QUANT	No	1.99M	2.08M	48M	19K	1.80K
	Yes	1.45M	1.48M	17M	9.3K	1.52K
Avg.	-	1.46M	1.45M	65M	20K	0.67K

Table 3.4 summarizes the simulation speeds of cycle-level models as compared to those of pure source-level, RTL or gate-level simulations. As discussed before, the CD-DT models are on average slightly slower than the CS-DT models. Compared to a pure source-level simulation, the CD-DT models are on average 45x slower. They are, however, about 73x and 2,200x faster



(a) DCT simulation.



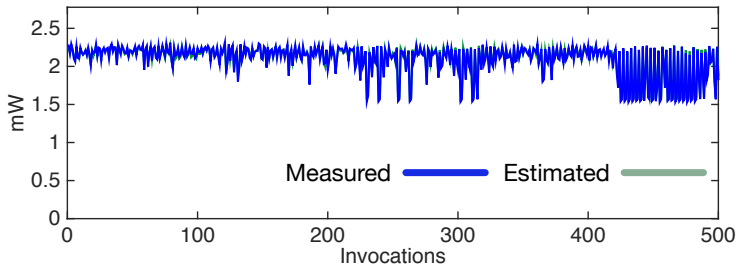
(b) HDR simulation.

Figure 3.7: Cycle-by-cycle power traces of cycle-level power model.

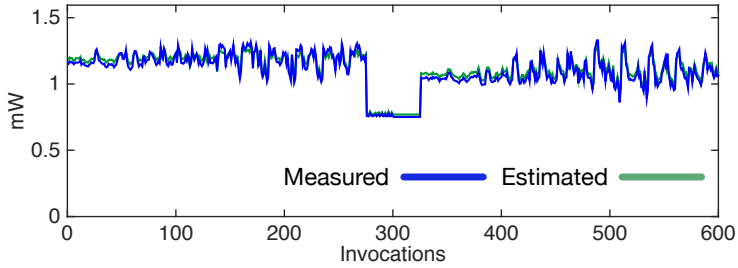
than RTL and gate-level power simulation, respectively.

Figure 3.7 and 3.8 show the cycle-by-cycle and invocation-by-invocation profiles of estimated versus measured power waveforms for the pipelined DCT and HDR designs. As the profiles show, our extended models are 100% timing accurate and can accurately track cycle-level power behavior within each invocation as well as data-dependent effects across different invocations of the same design.

The major learning overhead is collecting gate-level simulation results to construct the training vectors. Depending on the trace length and design complexity, we were able to generate gate-level power traces for training within



(a) DCT simulation.



(b) HDR simulation.

Figure 3.8: Invocation-by-invocation power traces of cycle-level power model.

6 to 20 minutes. The learning times of cycle-level models are proportional to the number of decomposed models, i.e. states. The synthesis time of cycle-level models takes 30 to 200 seconds. Overall, we were able to synthesize power models in each case within 24 minutes including trace generation.

Figure 3.9 further details the learning overhead and accuracy of the proposed decomposed model (CD) as compared to the single model (CS). By increasing the size of training sets, we explore trade-offs between learning overhead and final accuracy of trained models. We measure accuracy as cycle-by-cycle MAE of the models. All of these models utilize either the decision tree regression (-DT) or a least squares linear regression (-L). In all cases,

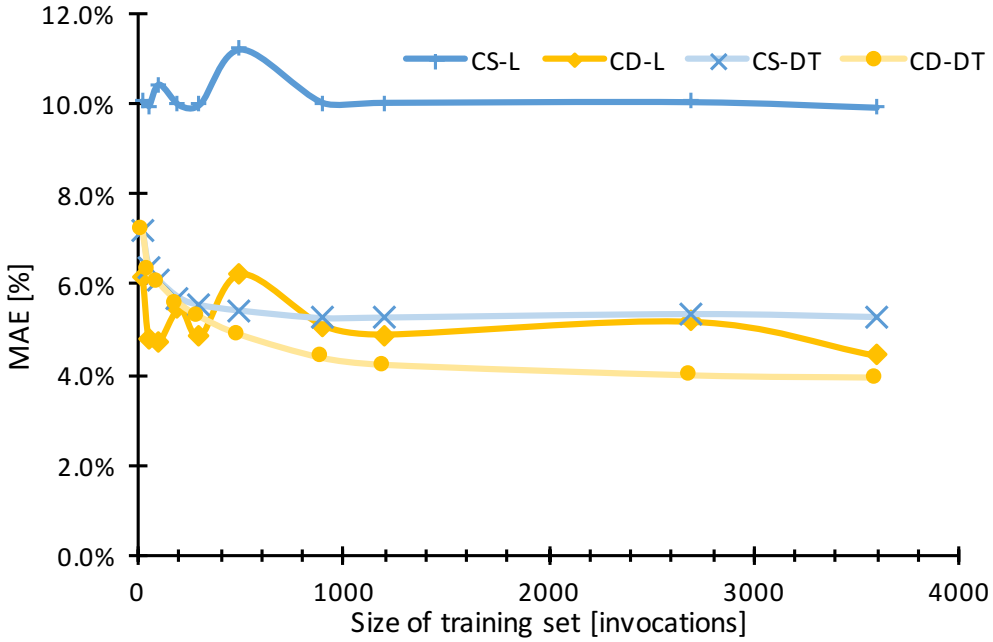


Figure 3.9: Learning overhead vs. cycle-level model accuracy for pipelined DCT.

the decomposed models show the better accuracy than the single models with sufficient training. The models with linear regression (CS-L and CD-L) suffer from overfitting trends and show worse accuracy than corresponding models with decision tree regression (CS-DT and CD-DT). We can observe that the cycle-level decomposed model utilizing the decision tree regression (CD-DT) provides the best accuracy for the same size of the training set, reaching more than 96% accuracy for a training set with 2,700 vectors.

3.4 Summary

In this chapter, we presented a novel approach for generating fast functional hardware models back-annotated with cycle-accurate and data-dependent power and performance estimates. Our back-annotation approach is fully automated by integrating with commercial off-the-shelf tools for custom hardware synthesized by high level synthesis. The proposed power model synthesis flow exploits structural scheduling and binding information to generate accurate and fast power models using advanced machine learning techniques. The proposed structural model decomposition enables accurate data-dependent power prediction while reducing simulation overhead. Associated activity models capture resource-level signal transitions without detailed full micro-architecture simulation. Our flow has been evaluated on several industry-strength benchmarks and generated models. Results show that our approach is able to achieve orders of magnitude speedup compared to gate-level or RTL power simulation, all while producing fully cycle-accurate timing results and estimating power with less than 10% cycle-by-cycle and less than 1% average error.

Chapter 4

Power Modeling for Gray-Box IPs

In the preceding chapter, we focused on enabling cycle-level, data-dependent power and performance estimation for high-level functional hardware models. In a white-box case, we support capturing cycle-accurate activity of RTL datapath resources, such as adders and multipliers, during high-level functional simulation by back-annotating abstract micro-architecture information into the IR. Resource-level tracing provides cycle-accurate switching activity of each datapath component, but requires extending the functional model to capture cycle-specific activities, resulting in simulation overhead.

In this chapter, we introduce an intermediate gray-box approach that supports power estimation at basic block-level granularity [53]. It utilizes less total switching activity and fewer invocations of the power model than cycle-level models, while providing a finer granularity than invocation-level models. The proposed approach first annotates limited mapping information about basic block inputs and outputs to extract block-level activity from a functional simulation. Instead of a cycle-level power model, we learn a block-level power model based on captured block-level activity and corresponding gate-level power traces. In this process, power models are decomposed into ba-

sic block specific models using control flow information to improve estimation accuracy.

The rest of the chapter is organized as follows: we first introduce proposed block-level annotation and activity computation in Section 4.1. Next, we propose the block-level power model synthesis in Section 4.2. Thereafter, we shows experimental results of applying the flow to a set of industrial-strength design examples in Section 4.3. Finally, Section 4.4 summarize the chapter.

4.1 Block-Level Activity Model

Internal signal switching activity estimation is a key for data-dependent power modeling. Resource-level tracing provides cycle-accurate switching activity of each datapath components, but requires extending the functional model to capture cycle-specific activities, resulting in simulation overhead. Moreover, Hamming distance and switching activity computation for whole resources is typically the most significant bottleneck for power estimation, and it is often much slower than actual functional simulation [54]. Instead of computing cycle-by-cycle switching activity for all resources, we propose a basic block-level model that only utilizes inter-basic block communication, i.e. inputs and outputs of basic block for activity and power estimation. This reduces the total amount of signal traces and switching activity that need to be collected and computed, which results in faster estimation speed.

4.1.1 Block-Level Annotation

In order to track inter-basic block communication activity in each block, we trace all input variables that are updated in a previous block but read in the current block, all output variables that are written in the current and read in a subsequent block, as well as all block-internal memory accesses. We extract the mapping of each input and output variable and each array access in the basic blocks to corresponding registers and memory ports in the hardware component taking into account register and memory port sharing. The annotation process inserts *trace*() function calls to store the inter-basic block communication traces along with the mapping IDs. Input variables are traced at the beginning of each basic block while output variables and memory access are traced at the end of each block.

4.1.2 Block-Level Activity Computation

Shared memory accesses can be flexibly scheduled during RTL synthesis to maximize hardware performance if there is no dependency between the operations. To compute accurate memory port activity, a reordering is therefore also required, but cycle-accurate reordering is not necessary. Instead of using a reordering buffer, the annotation process statically reorders the shared memory accesses by inserting the trace functions in access order.

Figure 4.1 shows the block-level activity computation process. Basic block inputs and outputs are collected in the trace buffer at run-time. At a beginning of the basic block, the trace functions attach basic block inputs to

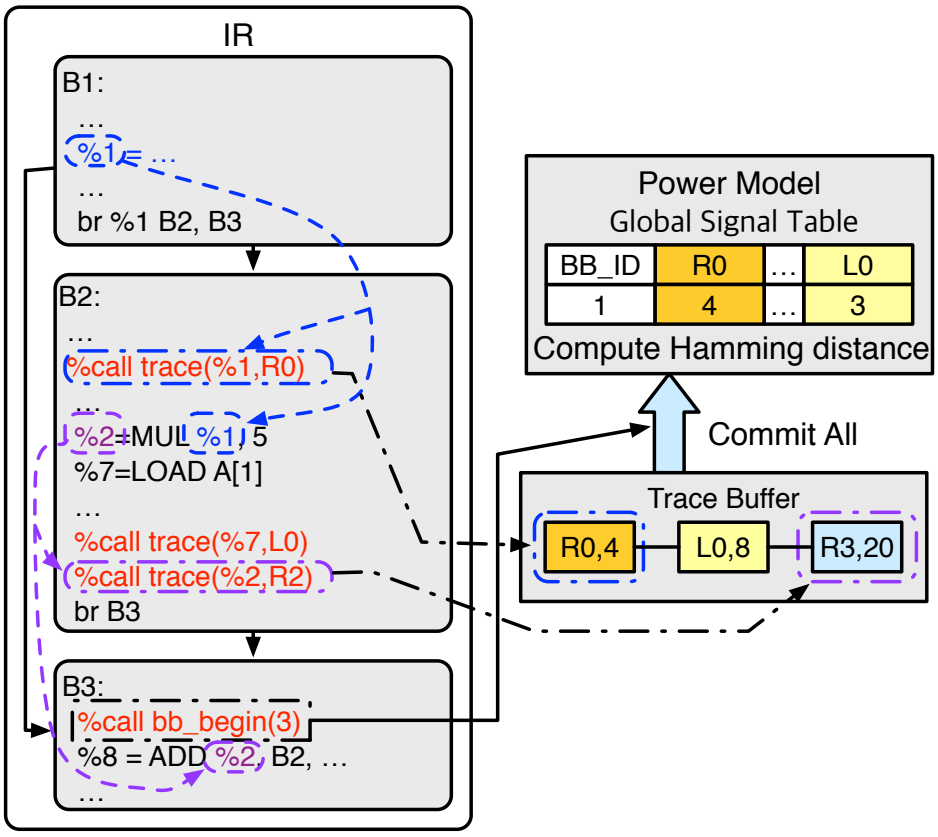
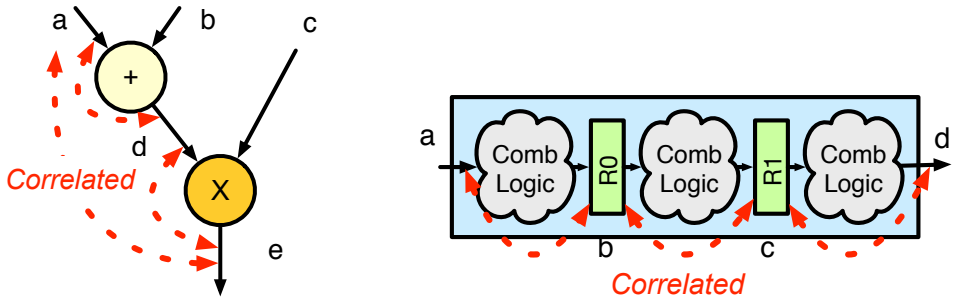


Figure 4.1: Basic block-level signal trace rearrangement.

the signal trace linked list, along with annotated resource information. At the end of each block, memory accesses and outputs are also attached to the linked list. At the beginning of the next basic block, the whole list of captured traces is committed into the global signal table to compute inter-basic block activity during a single basic block execution. For cycle-accurate performance estimation, the execution time of each basic block is also extracted from the HLS tools and annotated in a similar manner as for resource-level activity tracing.



(a) Combinational logic. (b) Multi-stage architecture.

Figure 4.2: I/O switching activity correlation.

4.2 Block-Level Power Model Synthesis

In the following, we describe block-level power model using only basic block inputs and outputs, including proposed power model decomposition.

4.2.1 Block-Level Power Modeling

Instead of internal resource-level activity, our block-level power model only utilizes switching activity of sampled basic block inputs and outputs for power estimation. Given the mapping of block inputs and outputs to registers or ports, internal signal activity in such an approach is indirectly observed from switching activity of input and output signals. Internal signal activity for pipelined and multi-stage hardware architectures in the current cycle can thereby be approximated from future and past switching activities of output and input registers/ports, respectively. We leverage the fact that internal switching activities are highly correlated with input and output activities. Figure 4.2(a) and Figure 4.2(b) show such correlations in combinational logic

and multi-stage architecture implementations, respectively.

The input and output switching activities of combinational arithmetic operators are linearly correlated [25]. Hence, the input switching activity of an operator can be modeled as a linear function of the input switching activity of the driving ancestor. For example, the power consumption of the dataflow graph in Figure 4.2(a) can be formulated as $p_n = \sum_{v=\mathrm{a...e}} \theta_v \cdot a_{n,v}$. Using such a linear input-output relationship, we can simplify this equation to $p_n = \sum_{v=\mathrm{a,b,c,e}} \theta''_v \cdot a_{n,v}$.

For pipelined or multi-stage architectures, input activity and activity of the first pipeline stage register are also linearly correlated. Similarly, activity of the second stage is linearly correlated to activity in the first stage. We can therefore approximately estimate internal switching activities throughout the pipeline from the input activity history. However, the activity of registers far away from the input are weakly correlated or not correlated at all. Instead, they are more likely to be correlated to activity at the outputs of the pipeline. Hence, to handle deeply pipelined logic and improve accuracy, we also consider future output activities for prediction. For a given pipeline of depth d, we can derive an I/O-based cycle-level power model P_{CI} as

$$\begin{aligned} p_n &= P_{\mathrm{CI}}(\mathbf{a}_{n-d+1,I}, ..., \mathbf{a}_{n,I}, \mathbf{a}_{n,O}, ..., \mathbf{a}_{n+d-1,O}) \\ &= \textstyle\sum_{i=0}^{d-1} \boldsymbol{\theta}''_i \cdot (\mathbf{a}_{n-i,I}, \mathbf{a}_{n+d-i-1,O}), \end{aligned} \tag{4.1}$$

where $\mathbf{a}_{n,I}$ and $\mathbf{a}_{n,O}$ denote the input and output activity vectors, and $\boldsymbol{\theta}''_i$ denotes coefficient vectors corresponding to pipeline stage i. For example, the power consumption of the pipelined hardware implementation in Figure 4.2(b)

can be computed as $p_n = \sum_{v=\mathrm{a,b,c,d}} \theta_v \cdot a_{n,v}$. Using I/O history and future, we can instead re-formulate power consumption as $p_n = \sum_{i=0}^{2} \boldsymbol{\theta}''_i \cdot (a_{n-i,a}, a_{n+2-i,\mathrm{d}})$. The power consumption of the micro architecture in Figure 3.4 can similarly be formulated as $p_n = \sum_{i=0}^{1} \boldsymbol{\theta}''_i \cdot (a_{n-i,\mathrm{a}}, a_{n-i,\mathrm{b}}, a_{n-i,\mathrm{c}}, a_{n-i,\mathrm{d}}, a_{n+1-i,\mathrm{i}})$ using activity history of primary I/O ports 'a', 'b', 'c', 'd', and 'i'. Note that this model applies to power estimation in all cycles/states S_n, $n = 1...3$, i.e. the stage-wise decomposition here is different from the state-wise in (4).

A block-level power model can then be formulated to estimate an average power consumption per basic block using switching activity of basic block inputs and outputs. Given a set of basic blocks B, where the m-th executed basic block is defined as b_m, the average power consumption $\bar{p}_m$ of basic block b_m can be formulated from (4.1) as

$$\bar{p}_m = \frac{1}{\bar{L}_m} \sum_{n=n_m}^{n_m+\bar{L}_m-1} \sum_{i=0}^{d-1} \boldsymbol{\theta}''_i \cdot (\mathbf{a}_{n-i,I}, \mathbf{a}_{n+d-i-1,O}), \tag{4.2}$$

where n_m and $\bar{L}_m$ denote the start cycle time and execution cycles of the m-th basic block, respectively. To simplify the equation, we can remove the summations over the pipeline and execution cycles by introducing a new coefficient vector $\bar{\boldsymbol{\theta}}$ and thus define a single block-level power model P_{BS} in the following manner:

$$\bar{p}_m = \frac{1}{\bar{L}} \bar{\boldsymbol{\theta}} \cdot \bar{\mathbf{a}}_m = P_{\mathrm{BS}}(\bar{\mathbf{a}}_m), \tag{4.3}$$

where $\bar{L} = \max_m \bar{L}_m$ denotes the maximum execution cycles over all basic blocks and $\bar{\mathbf{a}}_m = (\mathbf{a}_{n_m+j,I}, \mathbf{a}_{n_m+j+d-1,O} | \, 1-\mathrm{d} \leqq j < \bar{L})$ denotes a concatenation

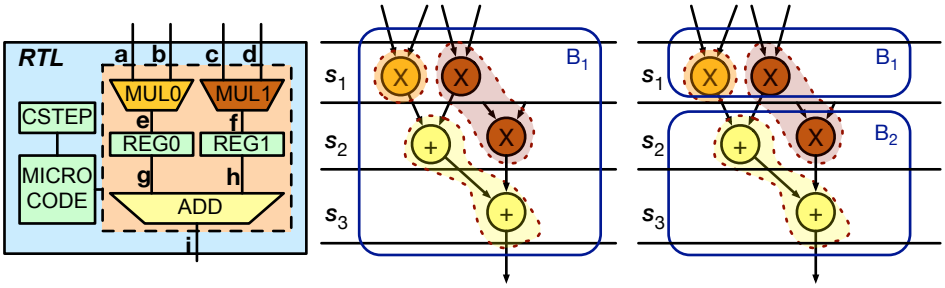


Figure 4.3: Block-level power modeling.

of all the activity of input and output ports that can flow through the pipeline for the length of the block. For blocks with length $\bar{L}_m < \bar{L}$, we zero pad vectors ($\mathbf{a}_{n_m+j,I} = \mathbf{a}_{n_m+j+d-1,O} = \vec{\mathbf{0}},\ \bar{L}_m < j < \bar{L}$) to keep the same dimension for all $\bar{\mathbf{a}}_m$.

4.2.2 Power Model Decomposition

In block-level gray-box activity models, we can only observe inputs and outputs of each basic block, not the actual cycle-by-cycle activity of all primary input and output registers or ports. For example, in Figure 4.3, 'a', 'b', 'c', 'd', and 'i' are primary I/O ports of the whole hardware. Assuming that a basic block starts and ends with S_1 and S_3, block inputs are 'a', 'b', 'c', and 'd' in state S_1, and 'c' and 'd' again in state S_2. Block output is 'i' once in S_3. We capture only these values, which are represented as different variables in the activity model code. Similarly, assuming the block starts with S_2, block inputs would be 'c', 'd', 'g', and 'h'. In this case, instead of using history of primary input ports for estimating activity of internal registers $REG0$ and $REG1$, we directly capture and trace block inputs 'g' and 'h'. In all cases, we

can therefore only use actual inputs and outputs of the current basic block to estimate the power consumption, where feature selection is implicitly applied to remove unnecessary signals not utilized in the block, thus reducing model complexity. With this, we can further decompose equation (4.3) into separate and independently learned block-specific power models P_{BD,b_m} for each basic block b_m in the following way:

$$\bar{p}_m = P_{\mathrm{BD},b_m}(\bar{\mathbf{a}}'_{m,b_m}) = \frac{1}{\bar{L}_m}\bar{\boldsymbol{\theta}}'_{b_m} \cdot \bar{\mathbf{a}}'_{m,b_m} \tag{4.4}$$

where $\bar{\mathbf{a}}'_{m,b_m}$and $\bar{\boldsymbol{\theta}}'_{b_m}$denote the block-level activity vector and corresponding coefficient vector for basic block b_m, respectively.

In case of pipelined or speculative scheduling, executions of successive basic blocks can overlap. Since we can not separate power consumption of overlapped blocks during training, we need to account for such periods by attributing power contributions of previous blocks that are still executing to the model of a current block. We redefine the execution length $\bar{L}_m$ of a characterized block b_m as the cycle difference between the start of its first operation and the start of the first operation of the next block. In other words, a block is defined to end when the next block starts. In addition, we extend the activity vector of a block by including activity vectors of all overlapping blocks. Such extended activity vectors may increase the complexity of the model, but only a small part of the transaction activities contribute to the power consumption in any given cycle, which results in many of the elements of the feature vector being zero or small. To prune away such uncorrelated features, we apply an additional feature selection for each decomposed model.

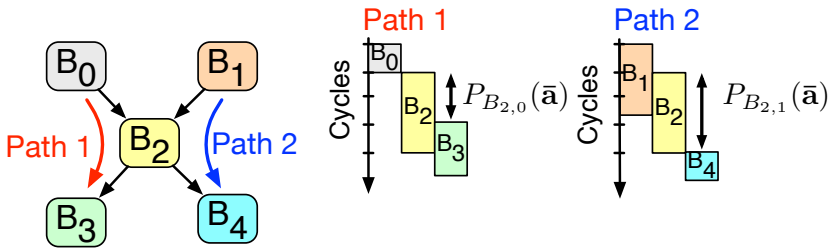


Figure 4.4: Basic block-level decomposition for multi-path control flow.

Finally, overlapped execution of blocks can also depend on control flow. Figure 4.4 shows the control flow graph of five basic blocks with two different paths (Path 1 and Path 2). Depending on the taken path, the length and overlapping of block B_2 varies. To account for such variations, we extract all possible unique combinations of predecessor and successor blocks that can overlap with each basic block during training. We then build different power models $P_{\mathrm{BD},b_{m,k}}$ for each possible unique overlappings k for a block b_m.

4.3 Experimental Results

We integrated block-level activity tracing and power model synthesis into our fully automated, HLS-based power modeling flow. We utilized the same benchmarks and experimental setup as described in the previous chapter (Section 3.3). Table 4.1 summarizes benchmarks and synthesis results including number of basic blocks in each design, the number of block I/O signals traced and the size of training and test sets.

Figure 4.5 and 4.6 show model accuracy and speed of proposed block-

Table 4.1: Benchmark summary for gray-box IPs

	Pipe	Basic Blocks	Traced Block I/O	Gates	Train Invoc.	Test Invoc.	Total Test Cycles
GEMM	No	10	4	703	2,000	5,000	3,670,000
	Yes	6	4	964	2,000	5,000	2,180,000
DCT	No	6	32	7,007	3,000	10,800	1,933,200
	Yes	6	32	6,309	3,000	10,800	1,015,200
HDR	No	13	24	4,883	988	1,200	1,194,000
	Yes	10	52	7,887	988	1,200	990,000
QUANT	No	6	10	1,032	3,600	12,288	7,150,452
	Yes	6	10	1,035	3,600	12,288	2,506,752

level decomposed power models (BD) as compared to the single basic block models (BS) across various benchmarks. To evaluate block-by-block MAE, we convert the gate-level cycle-by-cycle trace by assigning the average power dissipation to corresponding blocks. We compare both power models utilizing either a least squares linear regression (BS-L and BD-L) or a decision tree regression (BS-D and, BD-DT) against a decomposed model using a linear Bayes ridged (BD-BL) or a gradient boosting (BD-GB) regression. Decision tree based feature selection is applied in all cases.

The decomposed model using least squares regression (BD-L) shows up to 18% higher accuracy than a single model (BS-L). In case of pipelined QUANT and GEMM, accuracy is not improved since one single loop body block takes up most of the execution time. Using a decision tree regression (BS-DT) can similarly improve accuracy, but still shows higher errors in the complex cases (DCT, HDR). Among decomposed models, non-linear regression models (BD-DT and BD-GB) again show better accuracy than the linear ones

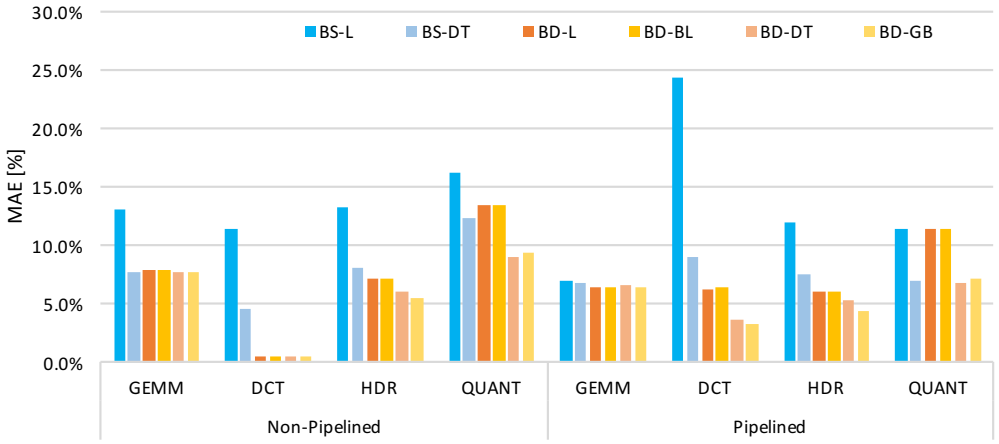


Figure 4.5: Basic block-by-basic block power accuracy.

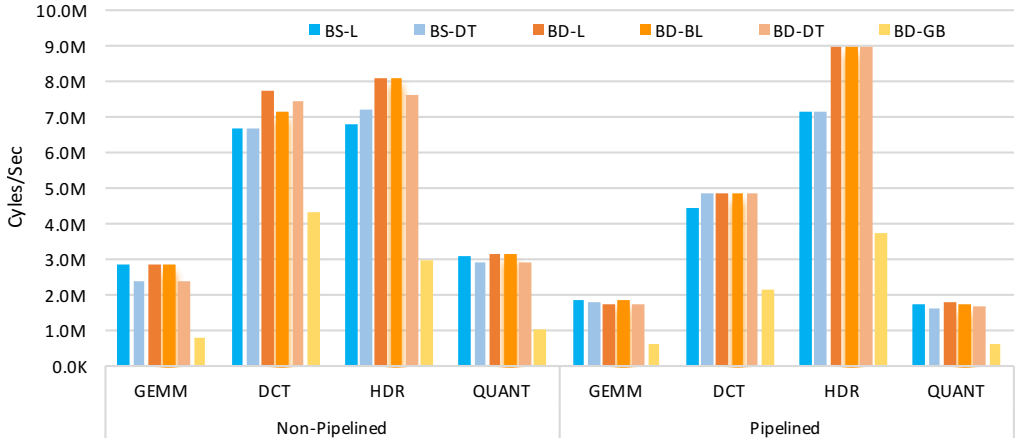


Figure 4.6: Estimation speed of models..

(BD-L, BD-BL), with up to 4.6% lower errors.

Figure 4.6 compares speed across various benchmarks. Here, decomposed models show faster estimation speed than single ones, since the latter require the union of all possible block inputs and outputs to be provided for each block. As before, the decision tree model (BD-DT) provides the best

Table 4.2: Accuracy of block-level modeling

	Pipe	Basic Block-by-Basic Block MAE [%]			Invocation-by-Invocation MAE [%]			Average MAE [%]		
		BS-DT	BD-DT	CD-DT	BS-DT	BD-DT	CD-DT	BS-DT	BD-DT	CD-DT
GEMM	No	7.8%	7.8%	7.9%	3.0%	3.0%	3.1%	0.4%	0.5%	0.4%
	Yes	6.8%	6.5%	6.5%	2.3%	2.2%	2.2%	0.1%	0.1%	0.1%
DCT	No	4.5%	0.5%	1.3%	3.5%	0.0%	0.0%	3.3%	0.0%	0.0%
	Yes	10.0%	3.6%	2.5%	5.9%	1.4%	1.1%	4.0%	0.2%	0.5%
HDR	No	8.0%	6.1%	3.2%	2.8%	1.9%	2.0%	1.9%	0.7%	0.9%
	Yes	7.6%	5.4%	3.1%	2.8%	1.9%	2.4%	2.3%	1.4%	1.0%
QUANT	No	12.3%	9.0%	10.0%	4.1%	3.0%	3.6%	1.6%	0.9%	0.1%
	Yes	6.9%	6.8%	6.0%	2.6%	3.0%	1.7%	0.4%	0.7%	0.4%
Avg.	-	8.0%	5.7%	5.1%	3.4%	2.1%	2.0%	1.8%	0.6%	0.4%

balance. It is on average almost as fast as linear models, and 2.3x faster than a gradient boosting (BD-GB) one at similar accuracy. Overall, block-level models provide similar accuracy than cycle-level estimates at significantly improved speed. The BD-DT model achieves on average 4.7Mcycles/sec at 94.3% accuracy.

Table 4.2 and Table 4.3 further summarize and detail accuracy and speed of models across benchmarks. We measure the data-dependent basic block-by-basic block MAE, invocation-by-invocation MAE, and total average error across a full simulation. We compare the block-level models (BS-DT, BD-DT) against our cycle-level decomposed model using decision tree regression (CD-DT). We can observe that the BD-DT models improve accuracy over the BS-DT models by a factor of 2x on average across all error metrics. Compared to the cycle-level model (CD-DT), the BD-DT models shows on average

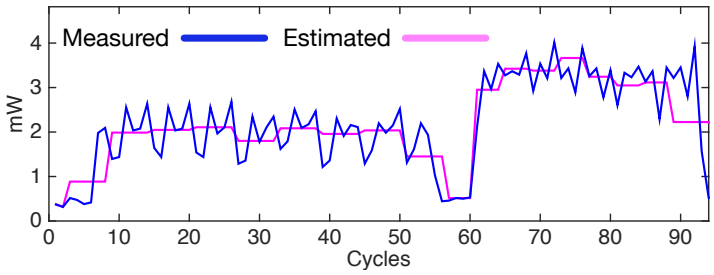
Table 4.3: Simulation speed of models [cycles/sec]

	Pipe	BS-DT	BD-DT	C Code	CD-DT	RTL	Gate
GEMM	No	2.34M	2.34M	220M	1.20M	51K	0.61K
	Yes	1.77M	1.70M	130M	0.84M	35K	0.36K
DCT	No	6.67M	7.44M	32M	1.40M	16K	0.41K
	Yes	4.83M	4.83M	17M	1.25M	5.9K	0.19K
HDR	No	7.19M	7.61M	32M	1.66M	13K	0.28K
	Yes	7.15M	8.94M	27M	1.65M	11K	0.20K
QUANT	No	2.87M	2.91M	48M	2.08M	19K	1.80K
	Yes	1.61M	1.64M	17M	1.48M	9.3K	1.52K
Avg.	-	4.30M	4.68M	65M	1.45M	20K	0.67K

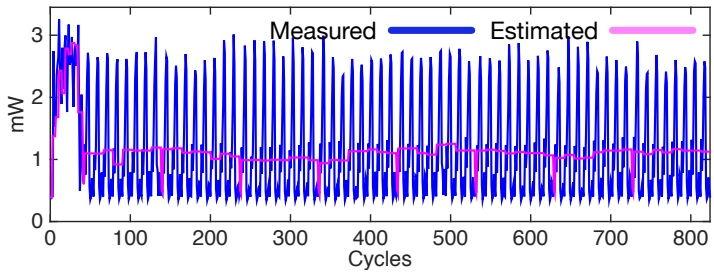
1.4x higher errors across all error metrics. Overall, the BD-DT models estimate block-level and invocation-level power consumption within 9.0% and 3.0% compared to gate-level power results, respectively. In all cases, average errors of the proposed BD-DT models across the whole simulation are below 1.4%.

Table 4.3 summarizes the simulation speeds of gray-box models as compared to those of pure source-level, cycle-level, RTL or gate-level simulation. We can observe that the BD-DT models are on average 1.1x faster than the BS-DT models. Compared to our cycle-level power model (CD-DT), the BD–DT models show on average 3x speedup. Overall, the BD-DT models are on average 15x slower then pure source-level simulation. However, they are about 220x and 6,500x faster than RTL and gate-level power simulation, respectively.

Figure 4.7 and 4.8 show the cycle-by-cycle and invocation-by-invocation profiles of estimated versus measured power waveforms for the pipelined DCT



(a) DCT simulation.

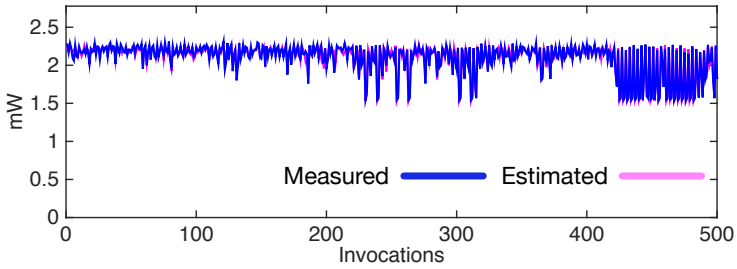


(b) HDR simulation.

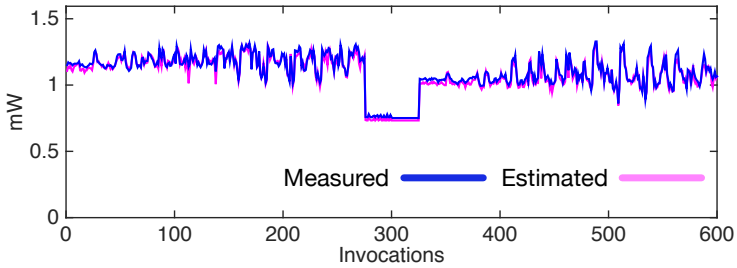
Figure 4.7: Cycle-by-cycle power traces of block-level power model.

and HDR designs. Note that the cycle-level trace of the block-level model shows the averaged power at block granularity. At the invocation-level, our extended models accurately track data-dependent effects across different invocations of the same design. All models show 100% accurately tracked timing of the hardware implementations.

As discussed in the previous chapter, the major learning overhead is constructing the training vectors using gate-level power simulation. As described before, we were able to generate gate-level power traces for training within 6 to 20 minutes. The learning times are proportional to the number of decomposed models, i.e. number of basic blocks. The synthesis time of



(a) DCT simulation.



(b) HDR simulation.

Figure 4.8: Invocation-by-invocation power traces of block-level power model.

block-level models takes 30 to 90 seconds. Overall, we were able to synthesize power models in each case within 23 minutes including trace generation.

Figure 4.9 shows the learning overhead and accuracy of the proposed decomposed model (BD) as compared to the single model (BS). We measure accuracy as basic block-by-basic block MAE of the models while increasing the size of training sets. All of these models utilize either a a least squares linear (BS-L, BD-L) or decision tree regression (BS-DT, BD-DT). In all cases, the decomposed models show better accuracy than the single models, which again indicates that the block-level decomposition improves the accuracy of the model. As has already been seen in cycle-level models, models utilizing

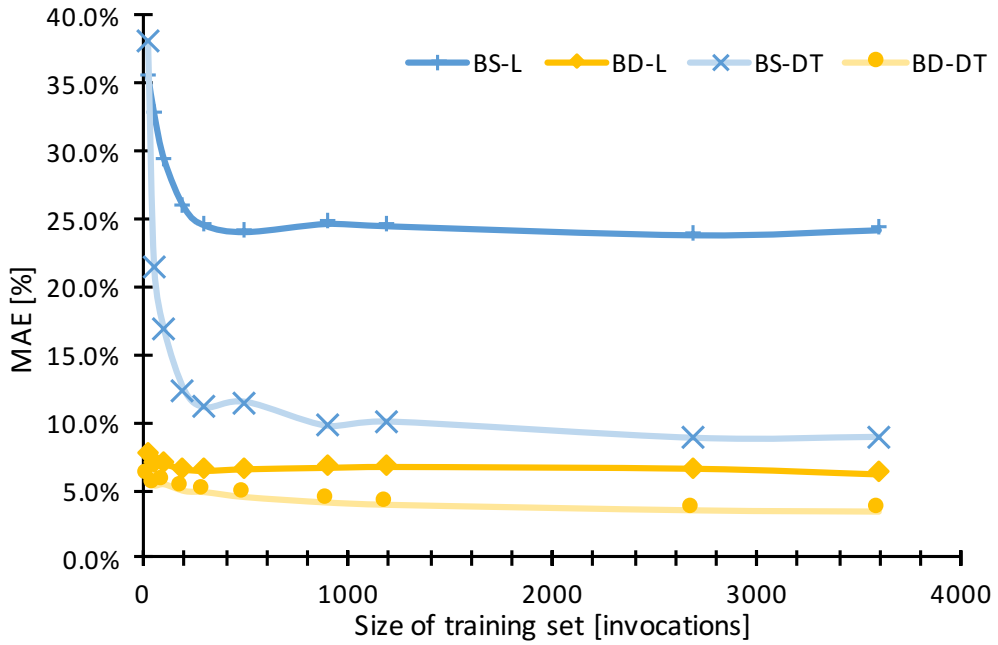


Figure 4.9: Learning overhead vs. block-level model accuracy for pipelined DCT.

decision tree regression always show better results than linear models. Overall, the block-level decomposed model utilizing the decision tree regression (BD-DT) provides the best accuracy for the same size of the training set, reaching more than 96% accuracy for a training set with 1,200 vectors.

4.4 Summary

In this chapter, we presented a power modeling approach for gray-box IPs. The proposed modeling flow extends functional hardware models to capture data-dependent block-level switching activity traces using block-level mapping information. The power model synthesis flow then exploits

basic block-level dataflow and control information to generate data-dependent power models at basic block-level granularity using advanced machine learning techniques. Experimental results demonstrate the accuracy and speed of the extended functional hardware models on several industry-strength benchmarks and generated models. Results show that the block-level power modeling approach is able to achieve 3x speedup compared to our cycle-level power model, all while estimating power within 9% basic block-by-basic block and 2% average error.

Chapter 5

Power Modeling for Black-Box IPs

A continued increase in system complexities has brought an increasing reuse of pre-designed hardware components acquired from third party vendors rather than being developed from scratch. Such IPs are not usually well documented, and only functional simulation models without detailed architecture descriptions are provided together with pre-synthesized gate-level implementations. This limited observability makes power modeling for such black-box IPs challenging. In the preceding chapters, we focused on enabling data-dependent power and performance estimation using full or partial hardware-internal information. In case of black-box IPs, such internal architecture information for fine-grain modeling is usually not available. This limits existing power estimation to coarse-grained simulation techniques using state-based models, which is inherently inaccurate.

In this chapter, we propose a novel power modeling approach for black-box IPs that is aimed at capturing accurate power consumption using state-of-the-art-machine learning techniques. We extract a data-dependent, invocation-level power model from gate-level cycle-by-cycle power traces. A given gate-level model and corresponding TLM of a black-box IP are simulated with the

same input vectors. Power synthesis then utilizes data I/O and control signal traces from TLM simulation together with cycle-level power traces from gate-level estimation to learn a power model. Based on the captured traces, we extract an invocation-by-invocation power model that enables fast yet accurate fine-grain data-dependent power estimation. Instead of building a single invocation-by-invocation model, the synthesis flow decomposes power models into multiple models and individually trains them. In the prediction phase, the decomposed models are combined into an ensemble estimation that predicts invocation-level power traces based on transaction-level I/O activity vectors with increased accuracy.

The rest of the chapter is organized as follows: we first propose the invocation-level activity model generation in Section 5.1. Next, we describe the details of our invocation-level power model synthesis in Section 5.2. Thereafter, Section 5.3 presents accuracy and speed of the proposed power models with a set of industrial-strength design benchmarks. Finally, Section 5.4 concludes the chapter with a summary.

5.1 External I/O Activity Computation

If no internal architecture information is available, we perform black-box power estimation utilizing only external I/O activity captured for each function invocation. In this case, the internal signal activity is indirectly observed from switching activity of input and output signals. As described in Chapter 4, Section 2.1, internal signal activity for pipelined and multi-stage

hardware architectures in the current cycle can be approximated from future and past switching activities of external output and external input ports, respectively. In a system-level model, re-arrangement of transactions and cycle-by-cycle I/O tracking is usually required to estimate cycle-level switching activity on input and output ports, which introduces significant simulation overhead. By contrast, our approach directly computes power estimates from unmodified high-level transaction-by-transaction activity. This approach can reduce tracing and computation overhead, but without internal timing information, only supports power modeling at invocation-level granularity.

5.1.1 External I/O Annotation

We assume that I/O interface mapping information between system-level transactions and the black-box data ports is given. System-level hardware models are usually written in system-level design languages (SLDLs), such as SystemC or SpecC. In such TLMs, communication interfaces are approximately modeled, and the detailed computation architecture is fully abstracted out. Models can also be purely functional, where no timing information is available. However, even a functional model has interfaces that map to corresponding data I/O ports. In general, we can find such mapping information in documents or test benches for gate level simulation.

We assume that data port mapping, bit widths and information about control signals is given, but internal architecture details are not available. Another assumption regarding observability in the system-level hardware model

is that some important control registers or control ports are available. Such control dependencies are also necessary to model functional or performance behavior. The activities of control signals, such as mode selections, do not by themselves affect power consumption. However, their value is utilized to estimate operating mode dependent power variations.

Required architecture information only consists of external I/O port mapping, bit width and control port/register information. Designers can manually describe the architecture file to utilize our automated annotation flow or manually insert trace functions into the source code. Both approaches can be seamlessly integrated into the automated power model synthesis process without further manual interventions.

5.1.2 External I/O Activity Computation

Mapping information and external I/O data are passed into annotated *trace*() calls, which are inserted at the beginning and end of each function. To compute I/O activity, we utilize a similar mechanism as at the basic block-level, but we commit signal traces into the global signal table only at the end of each function invocation. In addition to external data I/O activity, important control registers or control ports are also traced. We assume that such control dependencies are available to model functional or performance behavior. The activities of control signals, such as mode selections, do not by themselves affect power consumption. However, their value is utilized to estimate operating mode dependent power variations.

5.2 Invocation-Level Power Modeling

In the following, we describe our invocation-level power model utilizing external I/O activity, including proposed power model synthesis process.

5.2.1 Invocation-Level Power Model

As discussed in Chapter 4, we can formulate the power consumption of a hardware implementation using only past external input and future external output switching activities. The proposed invocation-level power model estimates an average power consumption per invocation using switching activity of such external I/O and control signals only. Given a per-invocation execution latency $\bar{\bar{L}}_l$ and assuming that the l-th invocation starts in cycle n_l, we can formulate an invocation-level power model P_{IC} that itself is not learned, but instead computes the average power $\bar{\bar{p}}_l$ of invocation l by averaging cycle-by-cycle power obtained from a single learned, I/O-based cycle-level power model P_{IC} according to (4.1) over the length of the invocation $\bar{\bar{L}}_l$:

$$\begin{aligned} \bar{\bar{p}}_l &= \tfrac{1}{\bar{\bar{L}}_l} \textstyle\sum_{n=n_l}^{n_l+\bar{\bar{L}}_l-1} P_{\mathrm{CI}}(..., \mathbf{a}_{n,I}, \mathbf{a}_{n,O}, ...) \\ &= P_{\mathrm{IC},\bar{\bar{L}}_l}(\mathbf{A}_{I,l}, \mathbf{A}_{O,l}). \end{aligned} \tag{5.1}$$

Here, $\mathbf{A}_{I,l}$ and $\mathbf{A}_{O,l}$ denote an external input and output activity matrix composed of $\bar{\bar{L}}_l$ input and output activity column vectors $\mathbf{a}_{n,I}$ and $\mathbf{a}_{n,O}$, $n_l \leqq n \leqq n_l + \bar{\bar{L}}_l - 1$, respectively. In this formulation, we assume that invocations do not overlap, and we enforce the following initial condition on the input and output activity vectors: $\mathbf{a}_{n,I} = \mathbf{a}_{n,O} = \vec{\mathbf{0}}$ for all $n < n_l$ or $n > n_l + \bar{\bar{L}}_l$.

In this model, re-arrangement of transactions and cycle-by-cycle I/O tracking is required to compute cycle-level switching activity on external input and output ports, which introduces a significant computation overhead. However, by fully expanding equation (5.1) following (4.1), it can be seen that invocation-level power does not actually depend on the order of activity information. Furthermore, if there is no transition in cycle n for input or output ports, the corresponding elements in external activity matrices $\mathbf{A}_{I,l}$ or $\mathbf{A}_{O,l}$ will be zero and terms will be masked. This indicates that we can formulate a single invocation-level power model P_{IS} by finding the contributed and reordered coefficients $\bar{\bar{\boldsymbol{\theta}}}$ purely from transaction-level activity vectors $\bar{\bar{\mathbf{a}}}$:

$$\bar{\bar{p}}_l = \frac{1}{\bar{\bar{L}}_l} \bar{\bar{\boldsymbol{\theta}}} \cdot \bar{\bar{\mathbf{a}}} = P_{\text{IS}}(\bar{\bar{\mathbf{a}}}). \tag{5.2}$$

We create the multiple such power models, one for each possible invocation latency $\bar{\bar{L}}_l$.

Transaction-level activity vectors are computed using Hamming distances over transaction data traces, where $\bar{\bar{\mathbf{a}}}$ is a concatenated vector composed over all transactions in an invocation, which does not require cycle-level rearrangement or cycle-by-cycle activity computation. However, the worst-case dimension of $\bar{\bar{\mathbf{a}}}$ is the product of the total number of external ports and execution cycles $\bar{\bar{L}}_l$, which may create generalization errors in learning processes.

5.2.2 Ensemble Model

To address complexity issues, we previously decomposed cycle- and block-level power models into separate and independent models for each state or block. However, this is not possible in black-box models, where control flow or state composition as well as scheduling and binding information is not available. However, since the current state is a function of the cycle n, we can indirectly capture the state based on n and an additional control vector $\mathbf{c}$. We thereby assume that control signals $\mathbf{c}$, if any, determine the IP operating mode on a per invocation basis, but remain constant over one invocation. With this, we can decompose the power model into separate and independently learned models $P_{\mathrm{ID},n}(\mathbf{c}, \bar{\bar{\mathbf{a}}})$ for each cycle n. In the process, we convert equation (5.2) into an ensemble of decoupled multiple regressions as follows:

$$\begin{gathered} \bar{\bar{p}}_l = P_{\mathrm{IE},\bar{\bar{L}}_l}(\mathbf{c}, \bar{\bar{\mathbf{a}}}) = \tfrac{1}{\bar{\bar{L}}_l} \textstyle\sum_{n=n_l}^{n_l+\bar{\bar{L}}_l-1} P_{\mathrm{ID},n}(\mathbf{c}, \bar{\bar{\mathbf{a}}}), \\ P_{\mathrm{ID},n}(\mathbf{c}, \bar{\bar{\mathbf{a}}}) = \bar{\bar{\boldsymbol{\theta}}}'_n \cdot (\mathbf{c}, \bar{\bar{\mathbf{a}}}), \end{gathered} \tag{5.3}$$

where $\bar{\bar{\boldsymbol{\theta}}}'_n$ denotes a decomposed coefficient vector and $(\mathbf{c}, \bar{\bar{\mathbf{a}}})$ the concatenation of control inputs and transaction activity. In (5.3), the dimension of each model $P_{\mathrm{ID},n}$ is the same as the single power model $P_{\mathrm{IS},\bar{\bar{L}}_l}$ from (5.2), i.e. the decomposed models use the complete transaction activity $\bar{\bar{\mathbf{a}}}$ at their input. However, only a small part of the transaction activities actually contribute to the power consumption in any given cycle. We leverage a decision tree based feature selection for each decomposed model to remove such unimportant features and reduce model complexity. As a result, the uncertainty of

the individual cycle-models is improved and there is less chance to run into generalization errors. Note that (5.3) is similar to (5.1), but (5.1) uses a single, uniform instead of separate and independent models for each cycle. Overall, the total number of models to learn is increased. However, each decomposed model uses the same input vectors, which enables parallel learning and prediction without additional overhead. Note that models could be further decomposed along control inputs. However, as the control space is exponential in the number of control signals, this would result in significant learning overhead.

The decomposition in (5.3) represents a form of ensemble learning. Ensemble learning is known to achieve better accuracy by utilizing the diversity over multiple learning models [55]. Traditional ensemble learning introduces diversity by dividing the training set, training each model with the partitioned training set, and then predicting the target value as the average over the prediction values of each model. By contrast, we introduce diversity by decomposing the model into separate cycle models.

Ensemble models are well known for providing better performance than single models in many cases [55]. In our case, we can prove that the proposed ensemble model in (5.3) shows better performance than the single invocation model $P_{\mathrm{IS}}(\bar{\bar{\mathbf{a}}})$. We can define the error-free perfect target function as $h(\bar{\bar{\mathbf{a}}})$. The sum-of-square errors of the single average model ($E_{P_{\mathrm{IS}}}$) can then be defined as

$$E_{P_{\mathrm{IS}}} = \mathbb{E}_{\bar{\bar{\mathbf{a}}}}[\{P_{\mathrm{IS}}(\bar{\bar{\mathbf{a}}}) - h(\bar{\bar{\mathbf{a}}})\}^2] = \mathbb{E}_{\bar{\bar{\mathbf{a}}}}[\varepsilon(\bar{\bar{\mathbf{a}}})^2], \tag{5.4}$$

where $\mathbb{E}_{\bar{\bar{\mathbf{a}}}}$ denotes the expectation with respect to the distribution of the input

activity vector $\bar{\bar{\mathbf{a}}}$, and control parameter is ignored for simplification. We further assume that a per-invocation execution latency $\bar{\bar{L}}_l$ is constant $\bar{\bar{L}}$. In the same way, the sum-of-squared error of the ensemble model ($E_{\bar{\bar{P}}_{\mathrm{IE}}}$) can be given by

$$\begin{aligned} E_{P_{\mathrm{IE}}} &= \mathbb{E}_{\bar{\bar{\mathbf{a}}}}\left[\{\tfrac{1}{\bar{\bar{L}}}\sum_{n=0}^{\bar{\bar{L}}-1} P_{\mathrm{ID},n}(\mathbf{c},\bar{\bar{\mathbf{a}}}) - h(\bar{\bar{\mathbf{a}}})\}^2\right] \\ &= \mathbb{E}_{\bar{\bar{\mathbf{a}}}}\left[\{\tfrac{1}{\bar{\bar{L}}}\sum_{n=0}^{\bar{\bar{L}}-1} \varepsilon_n(\bar{\bar{\mathbf{a}}})\}^2\right]. \end{aligned} \tag{5.5}$$

To simplify the problem, we assume that errors have zero mean and are uncorrelated,

$$\mathbb{E}_{\bar{\bar{a}}}[\varepsilon_m(\bar{\bar{\mathbf{a}}})] = 0, \quad \mathbb{E}_{\bar{\bar{\mathbf{a}}}}[\varepsilon_m(\bar{\bar{\mathbf{a}}})\varepsilon_k(\bar{\bar{\mathbf{a}}})] = 0, \quad m \neq k \ . \tag{5.6}$$

We further assume that all models are trained well and the sum-of-square errors of individual models are the same for simplification. We can obtain

$$E_{P_{\mathrm{IE}}} = \frac{1}{\bar{\bar{L}}} E_{P_{\mathrm{IS}}}. \tag{5.7}$$

Hence, the error of the ensemble model can be reduced by a factor of $\bar{\bar{L}}$ when the assumption that errors are uncorrelated is satisfied. In general, each decomposed model predicts a different cycle power, which implies that individual cycle errors will not be highly correlated. Moreover, we utilize non-linear learning approaches to prevent correlations between models, as will be discussed in the following section. As such, we can expect that the ensemble model always provides better accuracy than the single invocation one.

In the same manner, we can prove that the ensemble model has better accuracy than a single cycle-level power model P_{IC}. A single cycle-level

model estimates cycle-by-cycle power behavior using a single model instead of multiple decomposed ones. The sum-of-squared errors of the single cycle-level model can be formulated as equation (5.5). However, since each error is generated from the same model, i.e. is highly correlated, it cannot satisfy the assumption in (5.6). As a result, we can also expect that the ensemble model shows better prediction accuracy than a single cycle-level model.

5.2.3 Model Selection and Training

Each power model is trained from given power and activity traces. The activity traces collected from activity model simulation contain the cycles, decomposed model IDs, and corresponding switching vectors. Power traces contain actual power measurements from an equivalent gate-level simulation for the same set of training inputs. Activity and power traces are partitioned into model IDs, and then each power model is trained with the corresponding partitioned traces. Synthesized power models are thereby able to compute data-dependent power consumption estimates from the captured activity traces.

In general, a least squares linear regression over a set of training vectors has been widely employed to find the coefficient of power models. If there is a linear correlation between the power consumption trend and control data, equation (5.3) can be converted into a linear form. However, the control data for invocation models may have non-linear correlations with power consumption. To handle such problems, models could be further decomposed along

control inputs. By decomposing based on control signals, the power consumption behavior of each model could potentially become a linear function of the activity. The control data space, however, is exponential in the number of control signals, which results in significant learning overhead. Furthermore, as mentioned previously, power behavior of complex arithmetic units is generally correlated to Hamming distances of inputs and outputs, but not fully linear [25]. Moreover, linear regressions provide not enough diversity, which increases the error in the final ensemble model [56].

By contrast, depending on hardware functionality, input data statistics and complexity of models, a non-linear machine learning model can represent the power consumption behavior better than a typical linear least squares model while also providing more diversity, but this comes at the expense of estimation overhead. We thus evaluate various linear as well as non-linear regression models as part of our experiments.

5.3 Experimental Results

We implemented our annotation, power model synthesis and power prediction flow using the LLVM compiler framework [47], the scikit-learn [48] machine learning library and a natively implemented C++ online prediction library, respectively. We applied our flow to generate models for the benchmarks utilized in the previous chapters. Table 5.1 summarizes benchmarks and synthesis results including the number of traced external I/O ports, execution cycles per invocation, and the size of training and test sets.

Table 5.1: Benchmark summary for black-box IPs

	Pipe	Cycles per Invocation	Traced Ext. I/O	Gates	Train Invoc.	Test Invoc.	Total Test Cycles
GEMM	No	734	2/1	703	1,300	5,000	3,670,000
	Yes	436	2/1	964	1,300	5,000	2,180,000
DCT	No	179	4/4	7,007	3,000	10,800	1,933,200
	Yes	94	4/4	6,309	3,000	10,800	1,015,200
HDR	No	995	11/1	4,883	988	1,200	1,194,000
	Yes	825	11/1	7,887	988	1,200	990,000
QUANT	No	194	3/1	1,032	3,600	12,288	7,150,452
	Yes	68	3/1	1,035	3,600	12,288	2,506,752

Figure 5.1 and 5.2 compare model accuracy and speed of proposed invocation-level ensemble models (IE) as compared to averaged single cycle-level (IC) and single invocation-level (IS) power models across various benchmarks. We measured data-dependent invocation-by-invocation MAE of values predicted by each model compared to gate-level simulations. We compare all power models utilizing either a least squares linear (IC-L, IS-L, IE-L) or decision tree (IC-DT, IS-DT, IE-DT) regression against an ensemble model using a linear Bayes ridged (IE-BL) or a gradient boosting (IE-GB) regression. Decision tree based feature selection is applied in all cases.

The ensemble model using a least squares regression (IE-L) shows up to 6.2% and 1.4% lower errors compared to the single cycle- and invocation-level models (IC-L and IS-L), respectively. The non-pipelined DCT hardware shows large power variations in each cycle, but almost constant power consumption for each invocation. Since errors of the cycle-level model (IC-L) is generated from the same, single cycle model, they are highly correlated. As a result, the

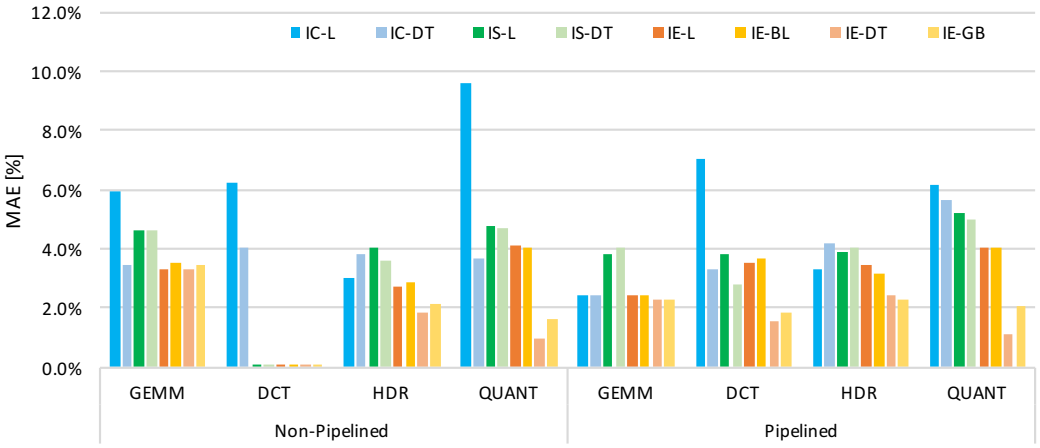


Figure 5.1: Invocation-by-invocation power accuracy.

error is not significantly reduced by averaging over invocations. The single invocation-level model (IS-L) shows better accuracy on average, but higher errors in complex cases utilizing long transaction activity vectors (GEMM, HDR). By contrast, the ensemble estimation (IE-L) utilizing decomposed cycle models does not suffer from such correlation and complexity problems. Utilizing decision tree regression provides on average 1.4x and 1.1x better accuracy for the single cycle- and invocation-level models (IC-DT and IS-DT), respectively, but ensemble model utilizing decision tree regression (IE-DT) show better accuracy in all benchmarks. Among ensemble models, linear regressions (IE-L and IE-BL) again show the worst accuracy. Non-linear models (IE-DT, IE-GB) show up to 3.3% additional accuracy improvement. The accuracy improvements in QUANT benchmarks are bigger than in other benchmarks due to the non-linear correlation between control inputs and power consumption. Overall, IE-DT and IE-GB estimate invocation-level power dissipation

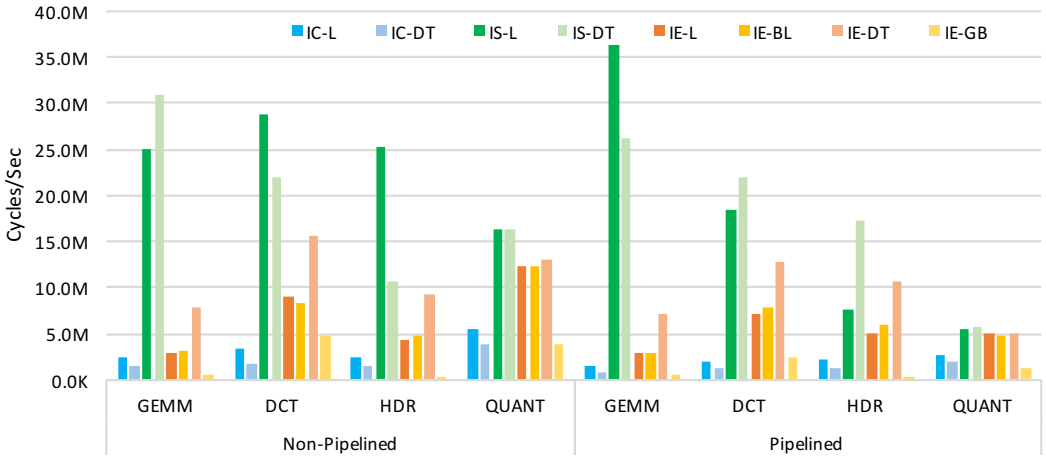


Figure 5.2: Estimation speed of models.

to within 3.3% MAE compared to gate-level power results.

When comparing speed (Figure 5.2), due to their simplicity, the single invocation model (IS-L and IS-DT) are on average significantly faster than others. Both the ensemble and cycle-level power models (IE-L and IC-L) estimate at a cycle-by-cycle level, but the ensemble models (IE-L) are on average 5x faster due to light-weight activity computation and parallelized cycle-level prediction. Among ensemble models, decision tree (IE-DT) models are the fastest. The dimension of activity features for the invocation-level model is much higher than resource- and block-level activity features. With such high-dimensional feature vectors, decision tree regressions can be faster than linear regression ones. Overall, when comparing different regression methods and models, results show that IE-DT provides the best trade-off between accuracy and speed. The IE-DT model achieves on average 10.19Mcycles/sec at 96.5% accuracy.

Table 5.2: Accuracy of invocation-level modeling

	Pipe	MAE			Avg. Error		
		IC-DT	IS-DT	IE-DT	IC-DT	IS-DT	IE-DT
GEMM	No	3.4%	4.6%	3.3%	0.4%	0.6%	0.5%
	Yes	2.4%	4.0%	2.3%	0.2%	0.1%	0.1%
DCT	No	4.0%	0.1%	0.1%	2.5%	0.0%	0.0%
	Yes	3.3%	2.8%	1.5%	1.3%	0.2%	0.2%
HDR	No	3.8%	3.6%	1.9%	0.2%	1.2%	0.7%
	Yes	4.2%	4.0%	2.4%	1.6%	2.1%	1.9%
QUANT	No	3.7%	5.1%	1.0%	2.3%	4.2%	0.2%
	Yes	5.6%	5.0%	1.1%	5.4%	5.1%	0.8%
Avg.	-	3.8%	3.6%	1.7%	1.7%	1.7%	0.6%

Table 5.2 and Table 5.3 further summarize and detail accuracy and speed of models across benchmarks. We measure data-dependent invocation-by-invocation MAE and total average error across a full simulation. Overall, the IE-DT models improve accuracy over the IS-DT and IC-DT models by a factor of 2.5x on average across all error metrics. IE-DT models estimate invocation-level power consumption to within 3% MAE and 15% maximum error compared to gate-level power results. In all cases, average errors across the whole simulation are below 2%.

Table 5.3 summarizes the estimation speeds of invocation-level models as also compared to a source-level, RTL, and a gate-level simulation. Overall, compared to an IS-DT and a pure source-level simulation, the IE-DT models are on average 2x and 6.3x slower, respectively. However, they are about 6x, 510x and 15,000x faster than an IC-DT, RTL, and gate-level estimation, respectively.

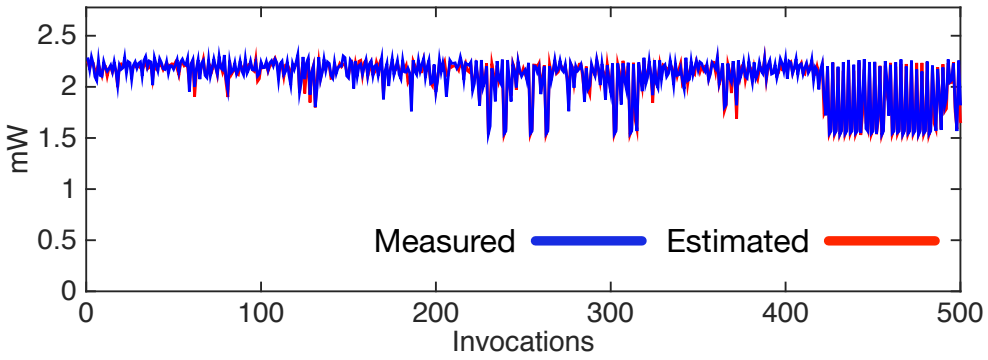
Table 5.3: Simulation speed of models [cycles/sec]

	Pipe	IC-DT	IS-DT	IE-DT	C Code	RTL	Gate
GEMM	No	1.57M	31.01M	7.92M	220M	51K	0.61K
	Yes	0.70M	26.16M	7.27M	130M	35K	0.36K
DCT	No	1.86M	21.97M	15.59M	32M	16K	0.41K
	Yes	1.17M	22.07M	12.69M	17M	5.9K	0.19K
HDR	No	1.54M	10.78M	9.17M	32M	13K	0.28K
	Yes	1.38M	17.30M	10.73M	27M	11K	0.20K
QUANT	No	3.97M	16.25M	13.00M	48M	19K	1.80K
	Yes	1.96M	5.70M	5.11M	17M	9.3K	1.52K
Avg.	-	1.77M	18.90M	10.19M	65M	20K	0.67K

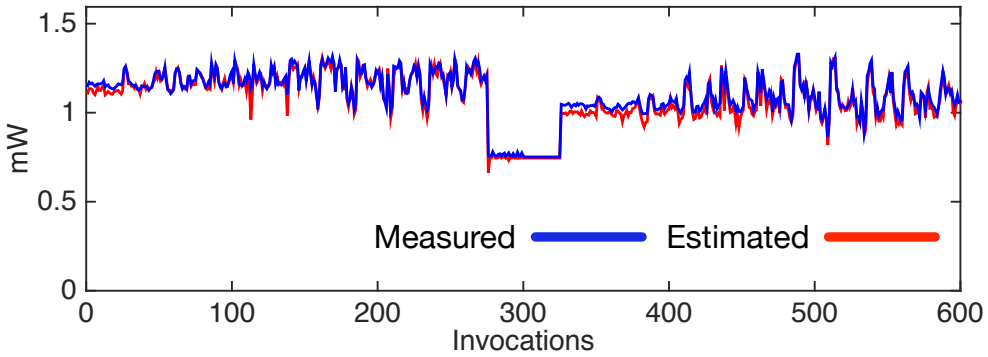
Figure 5.3 shows the invocation-by-invocation power traces of estimated versus measured power waveforms for the pipelined DCT and HDR benchmark designs using the ensemble model. As traces show, our synthesized models can accurately track data-dependent effects across different invocations of the same design.

In each case, we were able to synthesize power models within 24 minutes including trace generation for one-time gate-level simulation (which takes between 6 and 20 minutes). The learning times of the invocation-level models are proportional to the execution cycles per invocation. As mentioned previously, the invocation-level model supports parallel learning, which results in comparable learning speed to cycle-level models (around 30 to 200 seconds).

Figure 5.4 shows the learning overhead versus accuracy for the pipelined DCT benchmark. We compare the single cycle- and invocation-level models (IC and IS) with the proposed ensemble model (IE) utilizing either the decision



(a) DCT simulation.



(b) HDR simulation.

Figure 5.3: Invocation-by-invocation power traces of invocation-level model.

tree (-DT) or a least squares linear regression (-L).

In all cases, as in previous chapters, models utilizing decision tree regression always show better results than simple linear ones. Ensemble learning utilizing decision tree regression (IE-DT) provides the best accuracy for the same size of the training set, reaching more than 98% accuracy for a training set composed of 500 vectors. Ensemble and single-invocation models with linear regression (IE-L and IS-L) show similar accuracies, which indicates that

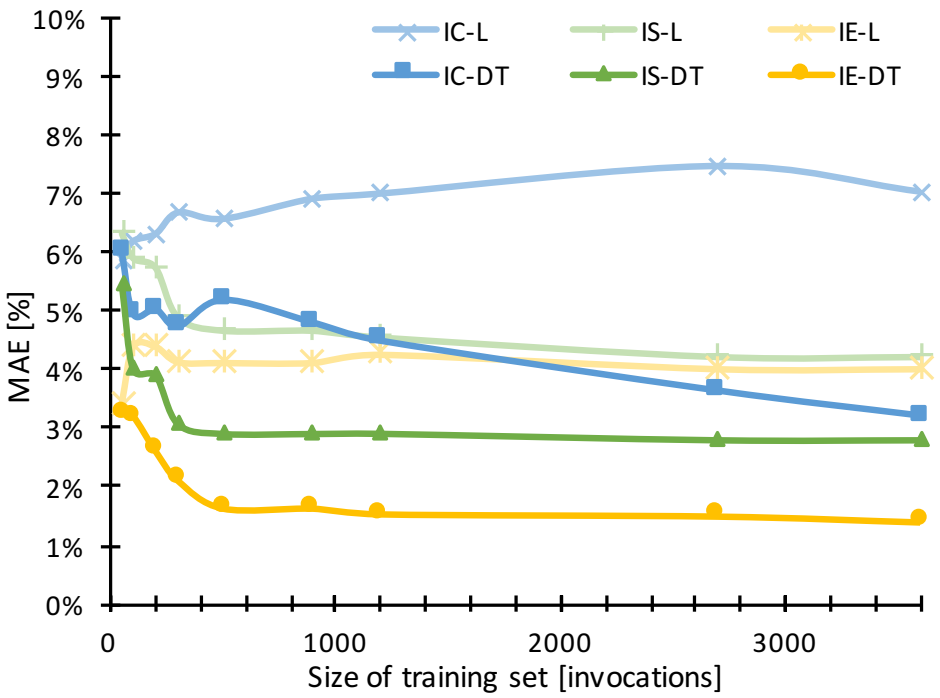


Figure 5.4: Learning overhead vs. invocation-level model accuracy for pipelined DCT.

individual linear regression models in the ensemble model are highly correlated and do not provide enough diversity. The single cycle model with least squares linear regression as similarly used in prior work and literature (IC-L) shows the worst accuracy with overfitting trends. As discussed in previous chapters, this again indicates that power behavior at the cycle level is inherently non-linear, especially when only being able to consider I/O activity.

5.4 Summary

In this chapter, we presented a novel approach for extending behavioral, transaction-level models of black-box hardware IPs with accurate power estimates. Our power model synthesis flow leverages state-of-the-art machine learning techniques to synthesize an invocation-level power model. The power model directly utilizes transaction-level external I/O activity and control information for fast estimation. The proposed model decomposition and ensemble estimation enable accurate data-dependent power prediction. Our flow has been evaluated on several industry strength benchmark designs and generated models. Results show that our proposed power model is able to achieve 6x faster prediction compared to cycle-level power models, and orders of magnitude speedup compared to gate-level power simulation, all while estimating power with less than 3% invocation-by-invocation and less than 2% average error.

Chapter 6

Overall Model Summary and Comparison

In this dissertation, we proposed three different high-level power modeling techniques for custom hardware IPs. In this chapter, we summarize model benefits across different levels by comparing accuracy, speed, and learning efficiency across different models and benchmarks.

6.1 Overall Speed and Accuracy Comparison

Table 6.1 and 6.2 summarizes accuracy and speed of models across benchmarks. We compare the accuracy of cycle-level decomposed models (CD), basic block-level decomposed models (BD), and invocation-level ensemble models (IE) utilizing decision tree regression in all cases. In addition to average errors across the whole simulation, we measure the data-dependent cycle-by-cycle, basic block-by-basic block, and invocation-by-invocation MAE predicted by each relevant model compared to gate-level simulations. We compute the basic block-by-basic block, and invocation-by-invocation errors of the cycle- and basic block-level models, respectively, by averaging power models over blocks and invocations. The cycle-level model (CD) shows better block-level accuracy than the basic block-level model (BD), similar invocation-level

Table 6.1: Summary of modeling accuracy

	Pipe	MAE						Average Error		
		Cycle	Basic Block		Invocation					
		CD	CD	BD	CD	BD	IE	CD	BD	IE
GEMM	No	10.1%	7.9%	7.8%	3.1%	3.0%	3.3%	0.4%	0.5%	0.5%
	Yes	7.9%	6.5%	6.5%	2.2%	2.2%	2.3%	0.1%	0.1%	0.1%
DCT	No	0.6%	1.3%	0.5%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
	Yes	3.9%	2.5%	3.6%	1.1%	1.4%	1.6%	0.5%	0.2%	0.2%
HDR	No	7.6%	3.2%	6.1%	2.0%	1.9%	1.8%	0.9%	0.7%	0.7%
	Yes	6.6%	3.1%	5.4%	2.4%	1.9%	2.4%	1.0%	1.4%	1.9%
QUANT	No	10.0%	10.0%	9.0%	3.6%	3.0%	1.0%	0.1%	0.9%	0.2%
	Yes	6.0%	6.0%	6.8%	1.7%	3.0%	1.1%	0.4%	0.7%	0.8%
Avg.	-	6.6%	5.1%	5.7%	2.0%	2.1%	1.7%	0.4%	0.6%	0.6%

accuracy to the ensemble model (IE), and the best average error across the whole simulation since it utilizes the largest amount of tracing. The BD model utilizes the smallest number of decomposed models, which results in the worst average prediction accuracy among all models. The ensemble approach (IE) utilizes the largest number of models, which enables a better invocation-level accuracy than others. Across all benchmarks, average errors of cycle-, block- and invocation-level models across the whole simulation are below 1%, 2%, and 2%, respectively.

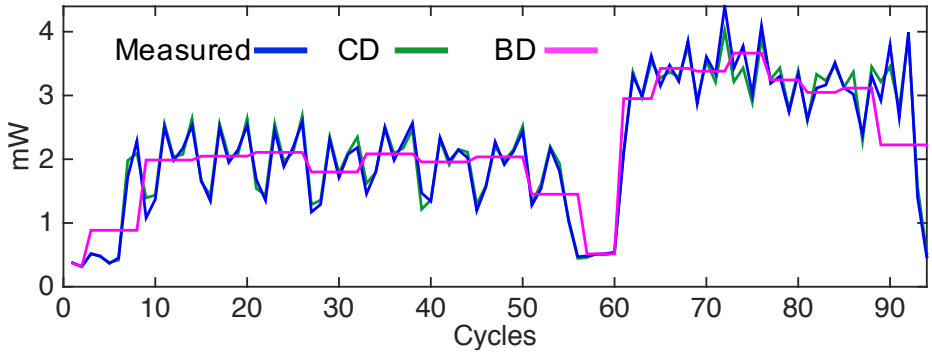
We also compare simulation speed of generated models to pure source-level, RTL, or gate-level simulation (Table 6.2). The block-level model (BD) is on average 3x faster than the cycle-level one (CD). This is due to its reduced activity features and smaller number of power model function calls. Both IE and CD utilize cycle-level estimation models, but the invocation-level

Table 6.2: Summary of simulation speed

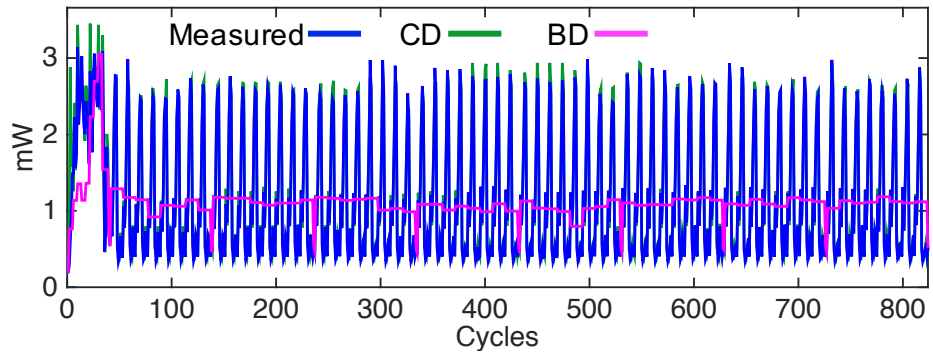
	Pipe	Speed [cycles/sec]					
		C Code	CD	BD	IE	RTL	Gate
GEMM	No	220M	1.20M	2.34M	7.92M	51K	0.61K
	Yes	130M	0.84M	1.77M	7.27M	35K	0.36K
DCT	No	32M	1.40M	6.67M	15.59M	16K	0.41K
	Yes	17M	1.25M	4.83M	12.69M	5.9K	0.19K
HDR	No	32M	1.66M	7.19M	9.17M	13K	0.28K
	Yes	27M	1.65M	7.15M	10.73M	11K	0.20K
QUANT	No	48M	2.08M	3.02M	13.00M	19K	1.80K
	Yes	17M	1.48M	1.66M	5.11M	9.3K	1.52K
Avg.	-	65M	1.45M	4.33M	10.19M	20K	0.67K

model is on average 7x faster. The light-weight activity computation and parallelized internal component power estimation of the I/O-based ensemble approach improves simulation throughput significantly. Overall, compared to a pure source-level simulation, the cycle-, block- and invocation-level models are on average 45x, 15x and 6x slower. However, they are about 73x, 220x and 510x faster than RTL power simulation, and 2,200x, 6,500x, and 15,200x faster than gate-level estimation.

Figure 6.1 and 6.2 show cycle-by-cycle and invocation-by-invocation profiles of estimated versus measured power waveforms for the pipelined DCT and HDR designs, respectively. As the profiles show, our proposed models can accurately track power behavior within each invocation, as well as data-dependent effects across different invocations of the same design.



(a) DCT simulation.

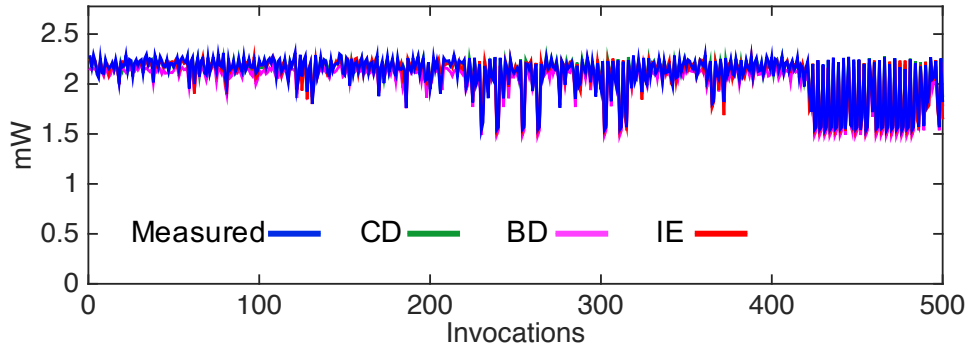


(b) HDR simulation.

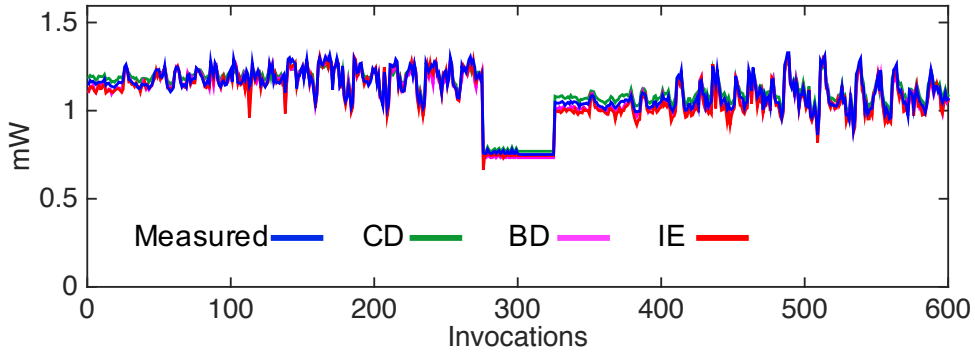
Figure 6.1: Cycle-by-cycle power traces for a single invocation.

6.2 Learning Overhead

As discussed before, the major learning overhead is collecting gate-level simulation results to construct the training vectors, where, depending on the trace length and design complexity, we were able to generate gate-level power traces for training within 6 to 20 minutes. The learning times of cycle-, block-, and invocation-level models are proportional to the number of decomposed models, i.e. states, basic blocks, and execution cycles per invocation, respectively. As mentioned previously, the invocation-level model



(a) DCT simulation.



(b) HDR simulation.

Figure 6.2: Invocation-by-invocation power traces.

supports parallel learning, which results in comparable learning speed to cycle-level models. The synthesis time of block-level models is the shortest with 30 to 90 seconds. Synthesis of cycle- and invocation-level models is on average three time slower than block-level models, taking 30 to 200 seconds. Overall, we were able to synthesize power models in each case within 24 minutes including trace generation.

Figure 6.3 further details the learning overhead and accuracy at different modeling levels for the pipelined DCT benchmark. We measure accuracy

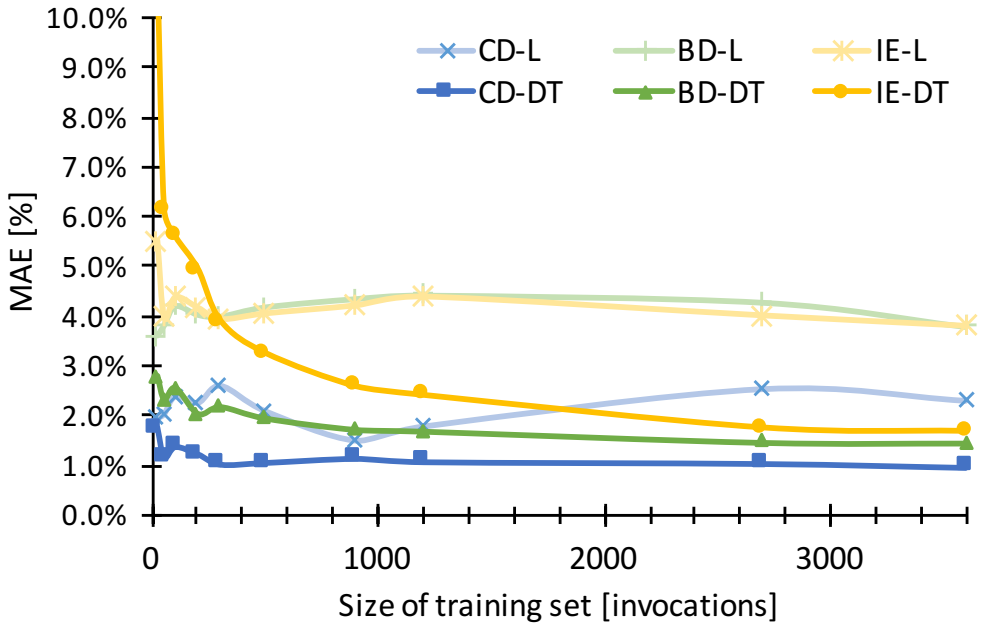


Figure 6.3: Learning overhead vs. model accuracy for pipelined DCT.

as invocation-by-invocation MAE of CD, BD and IE models utilizing either decision tree (-DT) or least squares linear regression (-L). In all cases, as discussed before, models utilizing decision tree regression always show better results than simple linear ones. Models with linear regression suffer from over-fitting trends, which indicates that IP power behavior is inherently non-linear. We can observe that the CD-DT model provides the best accuracy for the same size of the training set, reaching more than 99% accuracy for a training set with 300 vectors. The IE-DT model shows the worst learning efficiency, reaching 96% accuracy with the same amount of training. For the same training size, models based on more detailed and fine-grain estimation generally show better accuracy than more coarse-grain ones. Combined with opposing

trends in estimation speed, this establishes a trade-off between modeling level, accuracy, training efficiency and speed.

6.3 Summary

In this chapter, we compared accuracy, speed, and learning overhead of white-, gray-, and black-box power models. We can observe that there is a trade-off between observability of the model, accuracy, learning efficiency and estimation speed. Models using more detailed architecture information show the better accuracy for the same amount of training, but detailed activity tracing degrades simulation speed. Overall, our white-, gray-, and black-box power modeling approaches can predict cycle-, basic block-, and invocation-level power consumption to within 10%, 9%, and 3% of a commercial gate-level power estimation tool, respectively, all while running at several order of magnitude faster speeds of 1-10Mcycles/sec.

Chapter 7

Summary and Future Work

This chapter briefly reviews the dissertation and summarizes the contributions. Then, we discuss future research opportunities.

7.1 Summary

In this dissertation, we presented a comprehensive and fully automated framework that provides fast yet accurate learning-based power estimation of hardware IPs at three levels of abstraction.

We first introduced a power modeling for white-box hardware IPs. The proposed flow integrates with existing HLS tools to provide detailed FSMD-level architecture information of synthesized hardware implementations. An annotation process then refines the functional simulation model into an activity model, which captures data-dependent, cycle-accurate resource-level signal switching activity by annotating FSMD-level resource mapping information. Furthermore, we proposed a novel learning-based cycle-level power model synthesis approach, where we introduced a structural decomposition using scheduling and binding information to improve accuracy. Results show that using our white-box approach enables cycle-accurate and data-dependent

power estimation at speeds close to a functional simulation.

Detailed resource-level tracing provides cycle-accurate switching activity of each datapath component, but introduces simulation overhead. Instead of detailed cycle-level estimation, we further proposed a basic block-level model that only utilizes inputs and outputs of basic blocks for activity and power estimation. The proposed approach uses the mapping of each input and output variable and each array access in the basic blocks to annotate and trace activity of registers and memory ports in the hardware. Extended functional models are combined with a learning-based block-level power model to estimate basic block-by-basic block power consumption. Our gray-box model utilizes less total switching activity while also requiring fewer invocations of the power model, which results in faster estimation speed than our white-box approach with only a small loss of accuracy.

To apply either white-box or gray-box approaches, full or partial hardware-internal architecture information is required. However, black-box IPs usually provide only functional simulation models without detailed architecture descriptions together with pre-synthesized gate-level implementations. To provide power estimation for such black-box IPs, we finally proposed a novel power model that only utilizes external input and output history to track data-dependent pipeline behavior and drive a data-dependent, invocation-by-invocation power model extracted from gate-level cycle-by-cycle power traces. To synthesize the invocation-level power model, we proposed a specialized ensemble learning approach in which power models are decomposed into in-

dividual cycle-by-cycle models for efficient training and accurate prediction. Results shows that our black-box power model enables fast and data-dependent invocation-level power estimation without internal architecture information at only 6x slower speed compared to a pure high-level functional simulation.

By comparing all proposed models, we can observe trade-offs between observability, accuracy and estimation speed. Finer-grain architecture information can provide more accurate power models with the same amount of training, but also introduces activity tracing and prediction evaluation overhead.

In summary, this dissertation provides evidence that system-level functional models of custom hardware IPs can drive accurate fine-grain, data-dependent power models. This was achieved by annotating functional hardware descriptions with the ability to capture detailed hardware activity and by leveraging state-of-the-art machine learning technique with specialized model decompositions. We also showed that the proposed approach can be fully automated by integrating with existing, commercial high-level synthesis (HLS) tools for custom hardware synthesized by HLS.

7.2 Future Work

In our work so far, we have been able to develop an automated power modeling framework for the custom hardware IPs. In the following, we outline possible future research directions to extend proposed learning-based power modeling to embedded processors or accurate RTL power estimation.

7.2.1 Power Modeling for Embedded Processors

The increasing demand for wearable and portable Internet of Thing (IoT) devices with longer battery life has brought the need for low-power embedded processors and processor-based subsystems. In order to develop such energy efficient systems, fast and accurate system-level power estimation approaches are needed to drive associated optimization. However, a lack of fine-grain and accurate power models of processors renders power optimization of software, i.e. complete firmware or applications stacks running on such processors challenging and inaccurate.

To address this challenge, power modeling of embedded processors based on the custom hardware power modeling approaches developed in this work could be pursued. As an alternative to low-level power models driven by detail micro-architecture simulation, data-dependent and basic block-level power models could be combined and integrated with high-level abstract processor simulation frameworks, such as source-level or binary translated instruction set simulations. Based on the proposed white-box and black-box power estimation techniques, power models could be developed and trained by utilizing (dynamic) back-annotation of given processor micro-architecture activity information and/or extracted features from source and/or assembly code.

7.2.2 RTL Power Modeling with Structural Decomposition

Accurate RTL power estimation is important for many eariler design decisions. Existing learning-based approaches utilize feature selection methods

to collect signals or state variables highly correlated with power consumptions while managing the complexity of the power models. Such feature selection can reduce the model dimensions, but this may result in a loss of accuracy. By contrast, in this thesis, we introduced a structural decomposition approach for reducing power model dimensions that exploits scheduling and binding information to identify and remove unnecessary signals. We prove that a power model decomposition based on such structural micro-architecture information is able to reduce the complexity of the model with little to no information loss. As such, with structural decomposition, we anticipate to see improved accuracy and reduced learning overhead when applying similar concepts to RTL power estimation.

Bibliography

[1] J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, H. Huang, and G. Reinman, "Composable accelerator-rich microprocessor enhanced for adaptivity and longevity," in *Low Power Electronics and Design (ISLPED)*, Sep. 2013.

[2] F. Conti, C. Pilkington, A. Marongiu, and L. Benini, "He-P2012: Architectural heterogeneity exploration on a scalable many-core platform," in *International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, Jun. 2014.

[3] C. Pilato, P. Mantovani, G. D. Guglielmo, and L. P. Carloni, "System-level optimization of accelerator local memory for heterogeneous systems-on-chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 36, no. 3, Mar. 2017.

[4] I. Lee, H. Kim, P. Yang, S. Yoo, E.-Y. Chung, K.-M. Choi, J.-T. Kong, and S.-K. Eo, "PowerViP: SoC power estimation framework at transaction level," in *Asia and South Pacific Conference on Design Automation (ASP-DAC)*, Jan. 2006.

[5] C. Trabelsi, R. B. Atitallah, S. Meftali, J. luc Dekeyser, A. Jemai, C. Trabelsi, R. B. Atitallah, S. Meftali, J. luc Dekeyser, A. Je, H. I. Inria,

and A. Jemai, "A model-driven approach for hybrid power estimation in embedded systems design," *EURASIP Journal on Embedded Systems (EURASIP JES)*, vol. 1, Mar. 2011.

[6] S. Schürmans, D. Zhang, D. Auras, R. Leupers, G. Ascheid, X. Chen, and L. Wang, "Creation of ESL power models for communication architectures using automatic calibration," in *Design Automation Conference (DAC)*, May 2013.

[7] E. Copty, G. Kamhi, and S. Novakovsky, "Transaction level statistical analysis for efficient micro-architectural power and performance studies," in *Design Automation Conference (DAC)*, Jun. 2011.

[8] D. Lorenz, K. Grüettner, and W. Nebel, "Data-and state-dependent power characterisation and simulation of black-box RTL IP components at system level," in *Euromicro Conference on Digital System Design (DSD)*, Aug. 2014.

[9] S. Schürmans, G. Onnebrink, R. Leupers, G. Ascheid, and X. Chen, "ESL power estimation using virtual platforms with black box processor models," in *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, Jul. 2015.

[10] K. Grüttner, P. A. Hartmann, T. Fandrey, K. Hylla, D. Lorenz, S. Stattelmann, B. Sander, O. Bringmann, W. Nebel, and W. Rosenstiel, "An

ESL timing amp; power estimation and simulation framework for heterogeneous SoCs," in *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, Jul. 2014.

[11] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks, "Aladdin: A pre-RTL, power-performance accelerator simulator enabling large design space exploration of customized architectures," in *International Symposium on Computer Architecuture (ISCA)*, 2014.

[12] D. Chen, J. Cong, Y. Fan, and Z. Zhang, "High-level power estimation and low-power design space exploration for FPGAs," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan. 2007.

[13] N. R. Potlapally, A. Raghunathan, G. Lakshminarayana, M. S. Hsiao, and S. T. Chakradhar, "Accurate power macro-modeling techniques for complex RTL circuits," in *International Conference on VLSI Design (ICVD)*, 2001.

[14] L. Zhong, S. Ravi, A. Raghunathan, and N. K. Jha, "Power estimation for cycle-accurate functional descriptions of hardware," in *International Conference on Computer Aided Design (ICCAD)*, Nov. 2004.

[15] R. Tjarnstrom, "Power dissipation estimate by switch level simulation (cmos circuits)," in *International Symposium on Circuits and Systems (ISCAS)*, May 1989.

[16] T. H. Krodel, "Power play-fast dynamic power estimation based on logic simulation," in *IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD)*, Oct. 1991.

[17] C. X. Huang, B. Zhang, A.-C. Deng, and B. Swirski, "The design and implementation of PowerMill," in *International Symposium on Low Power Design (ISPLED)*, 1995.

[18] P. A. Beerel, C.-T. Hsieh, and S. Wadekar, "Estimation of energy consumption in speed-independent control circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 15, no. 6, Jun. 1996.

[19] G. Y. Yacoub and W. H. Ku, "An accurate simulation technique for short-circuit power dissipation based on current component isolation," in *International Symposium on Circuits and Systems (ISCAS)*, May 1989.

[20] S. M. Kang, "Accurate simulation of power dissipation in VLSI circuits," *IEEE Journal of Solid-State Circuits (JSSC)*, vol. 21, no. 5, pp. 889–891, Oct. 1986.

[21] N. L. W., "SPICE2 : a computer program to simulate semiconductor circuits," Ph.D. dissertation, University of California, Berkeley, May 1975.

[22] D. Kim, A. Izraelevitz, C. Celio, H. Kim, B. Zimmer, Y. Lee, J. Bachrach, and K. Asanović, "Strober: Fast and accurate sample-based energy sim-

ulation for arbitrary RTL," *SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 128–139, Jun. 2016.

[23] S. Ravi, A. Raghunathan, and S. Chakradhar, "Efficient RTL power estimation for large designs," in *International Conference on VLSI Design (ICVD)*, Jan. 2003.

[24] E. Macii, M. Pedram, and F. Somenzi, "High-level power modeling, estimation, and optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 17, no. 11, Nov. 1998.

[25] A. Bogliolo, L. Benini, and G. De Micheli, "Regression-based RTL power modeling," *ACM Transation on Design Automation Electronic System (TODAES)*, vol. 5, no. 3, Jul. 2000.

[26] V. Spiliopoulos, A. Bagdia, A. Hansson, P. Aldworth, and S. Kaxiras, "Introducing DVFS-management in a full-system simulator," in *International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOT)*, Aug. 2013.

[27] S. Gupta and F. N. Najm, "Power macromodeling for high level power estimation," in *Design Automation Conference (DAC)*, 1997.

[28] M. Barocci, L. Benini, A. Bogliolo, B. Ricco, and G. D. Micheli, "Lookup table power macro-models for behavioral library components," in *Alessan-*

dro Volta Memorial Workshop on Low-Power Design (VOLTA), Mar. 1999.

[29] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," in *International Symposium on Computer Architecture (ISCA)*, 2000.

[30] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "The McPAT framework for multicore and manycore architectures: Simultaneously modeling power, area, and timing," *ACM Transaction on Architecture Code Optimization (TACO)*, vol. 10, no. 1, Apr. 2013.

[31] G. Bernacchia and M. C. Papaefthymiou, "Analytical macromodeling for high-level power estimation," in *International Conference on Computer-Aided Design (ICCAD)*, Nov. 1999.

[32] H. Mehta, R. M. Owens, and M. J. Irwin, "Energy characterization based on clustering," in *Design Automation Conference (DAC)*, Jun. 1996.

[33] C. W. Hsu, J. L. Liao, S. C. Fang, C. C. Weng, S. Y. Huang, W. T. Hsieh, and J. C. Yeh, "PowerDepot: Integrating IP-based power modeling with ESL power analysis for multi-core SoC designs," in *Design Automation Conference (DAC)*, Jun. 2011.

[34] D. Sunwoo, G. Y. Wu, N. A. Patil, and D. Chiou, "PrEsto: An FPGA-accelerated power estimation methodology for complex systems," in *In-*

ternational Conference on Field Programmable Logic and Applications (FPL), Aug. 2010.

[35] G. Wu, "Performance, power, and confidence modeling of digital designs," Ph.D. dissertation, University of Texas at Austin, Aug. 2015.

[36] J. Yang, L. Ma, K. Zhao, Y. Cai, and T.-F. Ngai, "Early stage real-time soc power estimation using RTL instrumentation," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan. 2015.

[37] Y. H. Park, S. Pasricha, F. J. Kurdahi, and N. Dutt, "System level power estimation methodology with H.264 decoder prediction IP case study," in *International Conference on Computer Design (ICCD)*, Oct. 2007.

[38] ——, "A multi-granularity power modeling methodology for embedded processors," *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 19, no. 4, Apr. 2011.

[39] L. Kosmann, D. Lorenz, A. Reimer, and W. Nebel, "Enabling energy-aware design decisions for behavioural descriptions containing black-box IP-components," in *International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, Sep. 2013.

[40] Z. Zhao, A. Gerstlauer, and L. K. John, "Source-level performance, energy, reliability, power and thermal (PERPT) simulation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 36, no. 2, pp. 299–312, Feb. 2017.

[41] D. Calvo, P. González, L. Diaz, H. Posadas, P. Sánchez, E. Villar, A. Acquaviva, and E. Macii, "A multi-processing systems-on-chip native simulation framework for power and thermal-aware design," *Journal of Low Power Electronics (JOLPE)*, vol. 7, no. 1, pp. 2–16, 2011.

[42] C. Brandolese, W. Fornaciari, L. Pomante, F. Salice, and D. Sciuto, "A multi-level strategy for software power estimation," in *International Symposium on System Synthesis (ISSS)*, Sep. 2000.

[43] C. Brandolese, S. Corbetta, and W. Fornaciari, "Software energy estimation based on statistical characterization of intermediate compilation code," in *IEEE/ACM International Symposium on Low-power Electronics and Design (ISLPED)*, 2011.

[44] D. Lee, L. K. John, and A. Gerstlauer, "Dynamic power and performance back-annotation for fast and accurate functional hardware simulation," in *Design, Automation & Test in Europe (DATE)*, 2015.

[45] C. A. Ratanamahatana and D. Gunopulos, "Scaling up the naive bayesian classifier: Using decision trees for feature selection," *Applied Artificial Intelligence (AAI)*, vol. 17, 2003.

[46] Xilinx, "Vivado high-level synthesis," https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html.

[47] C. Lattner and V. Adve, "LLVM: A compilation framework for lifelong program analysis & transformation," in *International Symposium on Code*

Generation and Optimization (CGO), Mar. 2004.

[48] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research (JMLR)*, vol. 12, 2011.

[49] T. Mertens, J. Kautz, and F. V. Reeth, "Exposure fusion," in *Pacific Conference on Computer Graphics and Applications (PG)*, Oct. 2007.

[50] Synopsys, "Design Compiler," https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/design-compiler-graphical.html.

[51] Nangate, "NanGate FreePDK45 Open Cell Library," http://www.nangate.com.

[52] Synopsys, "PrimeTime," https://www.synopsys.com/implementation-and-signoff/signoff/primetime.html.

[53] D. Lee and A. Gerstlauer, "Learning-based, fine-grain power modeling of system-level hardware IPs," *under review*.

[54] M. Pedram, "Advanced power estimation techniques," in *Low power design in deep submicron electronics*, W. Nebel and J. Mermet, Eds. Kluwer Academic Publishers, 1997.

[55] C. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2007.

[56] M. Gashler, C. Giraud-Carrier, and T. Martinez, "Decision tree ensemble: Small heterogeneous is better than large homogeneous," in *Seventh International Conference on Machine Learning and Applications (ICML)*, Dec. 2008.

Vita

Dongwook Lee is a Ph.D. candidate at the University of Texas at Austin. He received the B.S. degree in electrical engineering from Seoul National University, Seoul, Korea in 2005, and the M.S. degree in electrical engineering and computer science from Seoul National University, Seoul. Korea, in 2007. His current research interests include system-level power modeling, system-level design automation, and system synthesis.

Permanent address: dongwook.lee@utexas.edu

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.