

RTL-aware Cycle-Accurate Functional Power Estimation

Lin Zhong *Student Member, IEEE*, Srivaths Ravi *Member, IEEE*, Anand Raghunathan *Senior Member, IEEE*, and Niraj K. Jha *Fellow, IEEE*

Abstract—Most methods for hardware power estimation operate at the register-transfer level (RTL), or lower levels of design abstraction. Since cycle-accurate functional descriptions (CAFDs) are being widely adopted in integrated circuit (IC) design flows, power estimation can potentially benefit from the inherent increase in efficiency of cycle-based functional simulation. However, attempts at power estimation for functional descriptions have suffered from poor accuracy because the design decisions performed during their synthesis lead to an unavoidable, large uncertainty in any power estimate that is based solely on the functional description.

We propose a methodology for CAFD power estimation that combines the accuracy achieved by structural RTL power estimation with the efficiency of cycle-accurate functional simulation. We achieve this goal by viewing a CAFD as an abstraction of a specific, known RTL implementation that is synthesized from it. We identify correlations between a CAFD and its RTL implementation, and “back-annotate” information into the CAFD solely for power estimation. The resulting *RTL-aware CAFD* contains a layer of code that instantiates virtual placeholders for RTL components, and maps values of CAFD variables into the RTL components’ inputs/outputs, thus enabling efficient and accurate power estimation. Power estimation is performed in our methodology by simply co-simulating the RTL-aware CAFD with a simulatable power model library that contains power macro-models for each RTL component. We present techniques to further improve the speed of CAFD power estimation, through the use of control state-based adaptive power sampling.

We have implemented and evaluated the proposed techniques in the context of a commercial C-based hardware design flow. Experiments with a number of large industrial designs (up to 1 million gates) demonstrate that the proposed methodology achieves accuracy very close to RTL power estimation with two-to-three orders of magnitude speedup in estimation times.

I. INTRODUCTION

Cycle-accurate functional descriptions (CAFDs) are commonly used for specification, efficient simulation, validation, and architectural exploration of hardware in systems-on-chip (SoCs). The emergence of C-based hardware description languages (HDLs), *e.g.*, SystemC [29], SpecC [28], and extensions to conventional HDLs, *e.g.*, System Verilog [30], to support specification at higher levels of abstraction than RTL, attests to this trend. The unmapped RTL style of the proposed Accellera standard for RTL semantics [1] describes paradigms for the use of CAFDs.

In order to support system-level design space exploration, and cope with increasing circuit complexities, it is natural to expect that power estimation techniques should also evolve to operate at higher levels of abstraction, where they could exploit inherent advantages such as increase in simulation

efficiency. However, this is not easily achieved, due to the loss of implementation details at higher levels of abstraction. Power is inherently a structural property, *i.e.*, estimating it requires a knowledge of the structural decomposition of the circuit into its constituent components (*e.g.*, RTL macro-blocks or logic gates), and input statistics (*e.g.*, switching activities) for each of them. CAFDs intentionally omit hardware structure for the sake of simulation efficiency. While accurate power estimation is usually possible at the structural RTL and lower levels, simulation at these levels is too slow. Therefore, a power estimation technique with the speed of functional simulation, yet with RTL-like accuracy, is desirable. Such a technique should also naturally plug into system-level simulation environments, and should provide detailed power information, *e.g.*, power breakdown over different parts of the circuit, or power variation over time.

A. Related Work

In order to provide feedback about power consumption at various stages in the design cycle, power estimation techniques have been developed that operate from the transistor level to the logic level and the RTL [7], [24], [26]. These technologies are relatively mature, and have been incorporated in commercial tools. We describe techniques for RTL power estimation in greater detail, since they are closest in terms of abstraction level to CAFDs.

Since an RTL description is structurally defined or can be directly mapped into a structural form, power estimation for a circuit is typically performed by aggregating power estimates for its constituent RTL components, such as arithmetic and logic components, latches and registers, multiplexers, *etc.* [14], [26]. Extensive research has been performed on techniques to characterize implementations of individual RTL components and derive efficient, yet accurate, *macro-models* [2]–[4], [8], [11], [18], [21], [23], [25], [32]. We utilize cycle-accurate power macro-models for RTL components [11], [18], [23], [25], [32] in our work. Although RTL power estimation can be relatively efficient for designs of limited sizes, it becomes extremely slow for large designs, especially when a power *vs.* time profile is needed. The main reason is that RTL circuit simulation is slow.

A few approaches to functional (or behavioral) power estimation have been investigated [10], [13], [17], [26]. Such techniques analyze a functional description without any regard to the RTL implementation that it is synthesized into. Although they enjoy the advantage of being fast, they are much less accurate than RTL power estimation. Hence, their utility is limited to fairly coarse-grained design decisions, *e.g.*, comparing algorithmic alternatives. Also, obtaining a power *vs.*

time profile is hard for purely functional descriptions, since the specification usually lacks cycle accuracy. In effect, the accuracy of conventional functional power estimation approaches is bounded by the inherent variation in power consumption across the space of different alternative RTL implementations, which is often as high as 2-3X.

B. Contributions and Paper Overview

In this paper, we address the problem of power estimation for a CAFD when its corresponding RTL implementation is known. Many high-level synthesis tools take CAFDs as their input, or generate a CAFD as a by-product or intermediate product after scheduling. Moreover, synthesizing a CAFD into a structural RTL implementation is very efficient compared to the time-consuming task of RTL simulation.

We view the CAFD as an abstraction of a specific RTL implementation, used in its place for efficient power estimation. We propose a technique to analyze a CAFD and its corresponding RTL implementation, and back-annotate information into the CAFD to enable higher power estimation accuracy. The resulting RTL-aware CAFD is simulated, together with power model libraries of various RTL components, to perform power estimation. We demonstrate that this approach enables power estimation accuracy (including spatial and temporal resolution) that is very close to RTL power estimation, at a speed that is comparable to cycle-accurate functional simulation. We believe that this combination of accuracy and efficiency is significant, and to our knowledge has not been achieved before.

While the basic idea of back-annotation has been extensively used in other contexts within the hardware design flow (*e.g.*, interconnect parasitics extracted from a layout are back-annotated into a gate-level netlist for accurate timing and power estimation), the back-annotation process for CAFDs is quite different. Our technique for back-annotation of information from RTL to CAFDs consists of the following steps: (i) virtual component (VC) instantiation, wherein virtual placeholders for RTL components are added to the CAFD purely for the purpose of capturing the relevant component input/output (I/O) values for power estimation, (ii) CAFD variable to RTL signal mapping, which allows the I/Os of virtual components to be updated during CAFD simulation, and (iii) idle cycle analysis, which addresses virtual components that are not explicitly activated in a simulation cycle but could dissipate power due to unnecessary changes in their inputs. The back-annotation process effectively adds a layer of code to the CAFD that maps the information available in the CAFD during simulation into the signal statistics needed for power estimation. While this obviously adds some simulation overhead, as demonstrated in our experiments, it is still much more efficient than simulating the entire RTL implementation.

In our investigation, we have found that the power consumption in many control states of a large circuit is highly predictable. We present techniques based on adaptive control state-based sampling, to further improve the speed of power estimation by optimizing the allocation of “computation effort” over time such that higher effort is expended during control

states for which power consumption displays a higher variation or less predictability.

We have prototyped the proposed techniques in the context of a commercial C-based high-level design flow [31], and applied them to large industrial designs (over 1M gates). Promising results (two-to-three orders of magnitude speedup, with about 2.0% average error and 4.3% cycle-by-cycle error), were achieved with respect to RTL power estimation.

The rest of this paper is organized as follows. We present background material and motivation for this work in Section II. We describe back-annotation techniques to produce RTL-aware CAFDs and the basic approach to CAFD power estimation in Section III. In Section IV, we motivate and describe the adaptive state-based sampling approach for further improving efficiency. We discuss the implementation of the proposed techniques in a commercial C-based design flow in Section V, which also provides experimental results on a number of industrial designs. We present conclusions in Section VI.

II. PRELIMINARIES

In this section, we discuss the issues involved in power estimation for CAFDs, and provide necessary background material.

A. Cycle-accurate Functional Descriptions

CAFDs accurately specify the behavior of a circuit for each cycle of its operation. Thus, from an I/O perspective, they are indistinguishable from structural RTL descriptions. CAFDs achieve simulation efficiency by omitting internal structural details of the circuit. For example, the user may be able to observe the values of only a subset of registers that are present in the implementation. In addition, they may not be bit-accurate, *i.e.*, they may use more efficient data types such as integers to replace bit-vectors when possible.

We focus on a popular class of CAFDs, called control state-based CAFDs, in which the design is represented as an extended finite-state machine (FSM), with functional descriptions for each state. Although the implementation of control state transitions may vary, the functional description of a state, or *state-functional description*, can be separated from the state-transition mechanism. Each *functional element* (operator, assignment, or variable reference) in a CAFD belongs to a unique state. Since each state is executed in a single clock cycle, a state-functional description has many special properties. For example, there must be no loop within it. If a variable/array element is the output of an operator, it cannot be an operand of any later operator in the same state-functional description, unless RTL components for these two operators can be chained.

Fig. 1(a) shows an example circuit that computes the greatest common divider (GCD) of two integers in the context of an SoC design. The functional description of GCD is given in a C-like language. For cycle-accurate simulation, the functional description of GCD is scheduled into a CAFD, as shown in Fig. 1(b). The CAFD is decomposed into control states, marked by ST_1, ST_2, and ST_3.

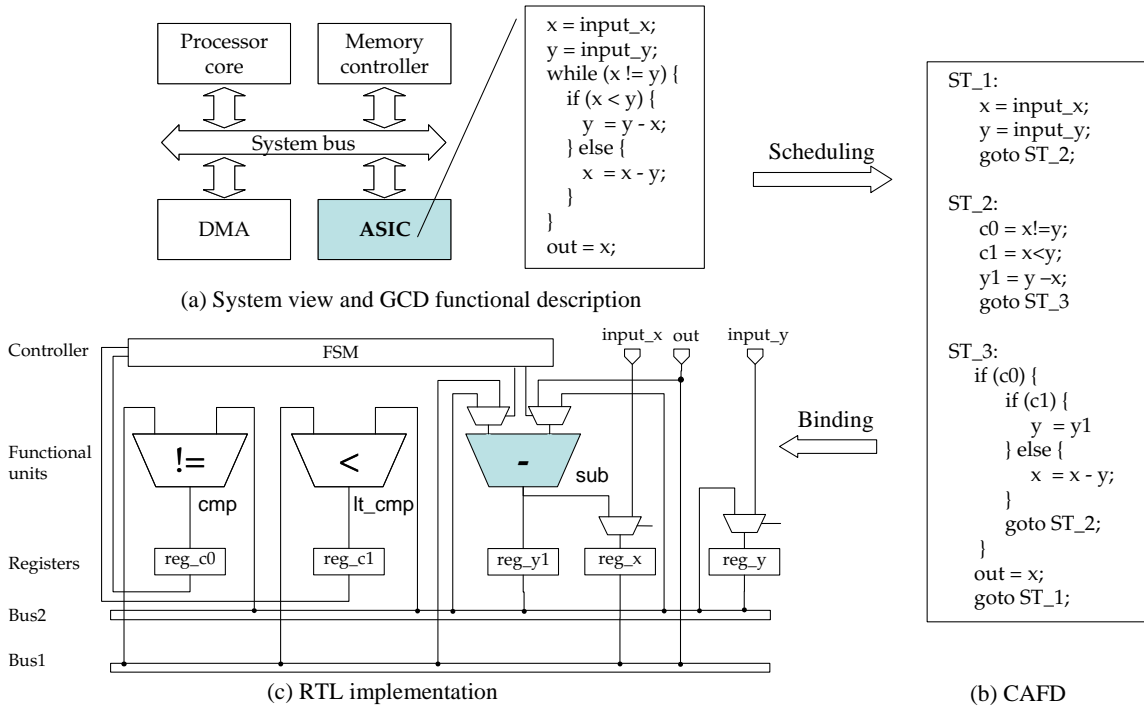


Fig. 1. Functional description, CAFD, and RTL implementation for the GCD example

B. From CAFDs to RTL

When a CAFD is synthesized into an RTL implementation, the synthesizer assigns functional elements to RTL components. While the synthesizer knows how a functional element is implemented in hardware, this knowledge is often discarded after synthesis. In our work, we extract this information and use it to enhance the CAFD for accurate power estimation. RTL components, such as registers, functional units, memories, and data-transfer interconnects, can be explicitly associated with functional elements in the CAFD. They are said to be *functionally-explicit*. Other RTL components that cannot be associated with any functional elements, *e.g.*, multiplexers, and control logic, are called *functionally-implicit*. If a functionally-explicit RTL element is active in a state, *i.e.*, one of the functional elements from that state are mapped to it, the values of its inputs and outputs can be obtained from the CAFD by tracing the appropriate variables. For example, consider the subtractor in an RTL implementation of the GCD circuit illustrated in Fig. 1(c). In state ST_2 in Fig. 1(b), the operation $y1 = y - x$ is mapped to the subtractor, which means that its inputs assume the values of variables y and x , and its output assumes the value $y1$.

C. Challenges in Power Estimation for CAFDs

It is hard to estimate power consumption based on the CAFD alone, since it does not specify the components utilized in the circuit. For example, the CAFD shown in Fig. 1(b) for the GCD example can be synthesized using either one subtractor or two, and using either one multi-function comparator or separate $<$ and $!=$ comparators. Furthermore, even if the

number of components in the implementation is fixed, the manner in which the operations and variables in the CAFD are mapped to components can affect power consumption. However, if an RTL implementation is supplied, information can be derived from it to enable CAFD power estimation. For example, for the RTL implementation shown in Fig. 1(c), we know that all subtraction operations are bound to the single subtractor (sub), shown in grey. This implies that, whenever the CAFD is in control state ST_2, the subtractor performs the operation $y1 = y - x$, and the inputs to the subtractor assume the values of CAFD variables y and x . Similarly, in state ST_3, the input values for the subtractor can be determined easily only when $(c0, c1)$ equals $(1, 0)$, *i.e.*, the subtraction operation $x = x - y$ is executed. If we were able to deduce the input values to each component in each CAFD state (equivalently, each simulation cycle), we could perform fairly accurate power estimation using power macro-models for each RTL component. Unfortunately, it is not clear what the input values of the subtractor are for state ST_1 or for ST_3 when $(c0, c1)$ does not equal $(1, 0)$. In the corresponding cycles, there are no CAFD operations mapped to the subtractor, *i.e.*, it is idle. In the idle cycles, the input values depend on how the multiplexers feeding the subtractor are configured, and the values at the selected data inputs.

Generalizing the observations from the above example, the following questions need to be addressed to solve the problem of accurate CAFD power estimation.

1. How can the minimum information necessary for power estimation be extracted from the RTL implementation?
2. How can this information be automatically back-

annotated into the CAFD?

- How can inputs for idle components in each control state be determined?

Answers to these three questions form the basis of our RTL-aware cycle-accurate functional power estimation technique.

D. Evaluating Power Estimation Accuracy

The objective of CAFD power estimation is to produce power estimates that are as close as possible to those obtained using a lower-level (RTL or gate-level) estimation tool. We next define the accuracy metrics used in our work.

Consider a circuit and an input test bench of N cycles. Let $P(i)$, $i = 1, 2, \dots, N$, denote the power consumption of the circuit in the i th cycle, as estimated by a reference power estimation tool (RTL or gate-level in our work). Let $P'(i)$ denote the power estimate for the i th cycle produced by the proposed CAFD power estimation technique. P_{avg} and P'_{avg} are the corresponding average power estimates over the entire test bench.

The average or accumulative power estimation error is given by

$$Avg.Error = \left| \frac{P'_{avg} - P_{avg}}{P_{avg}} \right| \cdot 100\%$$

Often, it is also useful to estimate the variation of power consumption over time. The absolute cycle power error (ACPE) for the i th cycle is defined as

$$ACPE(i) = \left| \frac{P'(i) - P(i)}{P(i)} \right| \cdot 100\%$$

The average ACPE (AACPE) over the N cycles is used to measure the accuracy of cycle-by-cycle power estimation. Note that power underestimations and overestimations in specific cycles will not cancel each other in the AACPE metric. Naturally, obtaining a low AACPE is more challenging than obtaining a low average power error as will be seen later on. In this work, $P(i)$ is obtained using a commercial RTL power estimation tool, while $P'(i)$ is produced by the techniques described in this paper.

III. CAFD POWER ESTIMATION METHODOLOGY

In this section, we discuss our power estimation methodology in detail.

Fig. 2 presents an overview of our methodology for CAFD power estimation. We are given a CAFD, its corresponding simulation test bench, and a power model library for RTL components. The library contains power macro-models for each type of RTL component, which express power consumption as a function of the current and previous input vectors. The power model library is generated once for each fabrication technology, using well-known characterization techniques [2]–[4], [8], [11], [18], [21], [23], [25], [32], and will not be further described here.

The CAFD is first preprocessed in order to facilitate back-annotation of RTL information, and subject to high-level synthesis to generate an RTL implementation. Alternatively, the CAFD may be generated as an intermediate by-product of

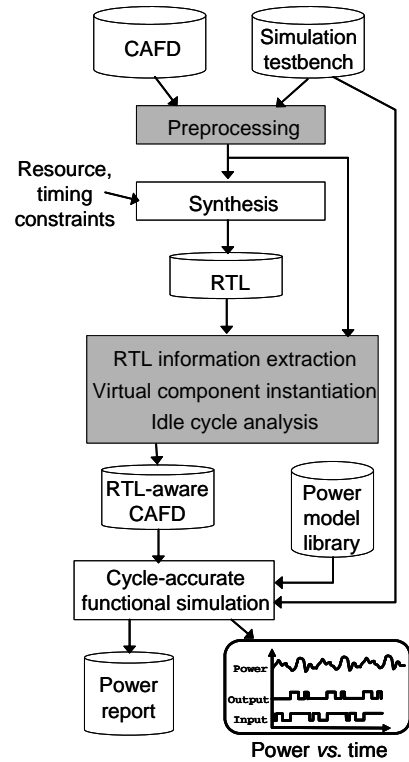


Fig. 2. Overview of the proposed CAFD power estimation methodology

high-level synthesis starting from a pure behavioral description. The preprocessed CAFD and RTL implementation are analyzed to derive the minimum necessary information, which is then back-annotated into the CAFD through virtual component instantiation and idle cycle analysis, resulting in an RTL-aware CAFD. The RTL-aware CAFD is co-simulated with the power model library under the given test bench to generate an average power report or power *vs.* time waveforms.

The structure of an RTL-aware CAFD is shown in Fig. 3. The enhancements performed to the original CAFD for power estimation are shown shaded in grey. The RTL-aware CAFD includes a “virtual component” for each component in the RTL implementation. The virtual components, which are automatically instantiated by our methodology, are simply placeholders to collect the information necessary to invoke the power model, *i.e.*, the component’s I/O values in the current and previous cycles. They do not simulate the actual functionality of the component they represent. Virtual components are also responsible for invoking the component power model during each simulation cycle, and storing the resulting power estimate for use in power aggregation and reporting. The RTL-aware CAFD also includes automatically generated I/O mapping code that maps the values of CAFD variables to the I/O values for virtual components. The power aggregation and reporting code sums up the power values from all the virtual components according to the circuit hierarchy, and keeps relevant statistics such as the power breakdown by component type. It is also responsible for generating the average power consumption report, or a power *vs.* time dump that can be viewed using standard waveform viewers.

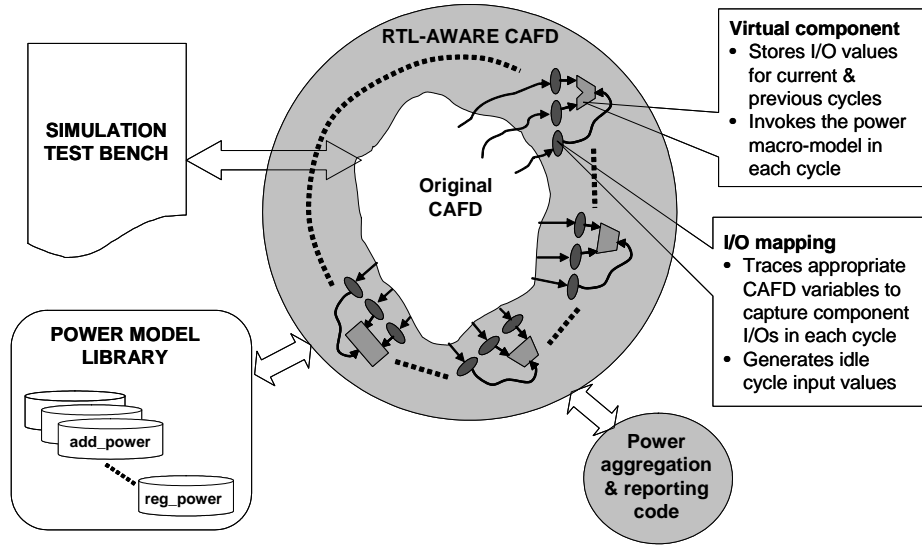


Fig. 3. Structure of an RTL-aware CAFD, and its use for power estimation

In the remainder of this section, we describe in detail the steps shaded in grey in the methodology of Fig. 2. We conclude the section with a discussion of the limitations and sources of error in our approach.

A. Preprocessing

To facilitate the back-annotation of RTL information into a CAFD, we preprocess the CAFD so that each functional element can be uniquely identified by the name and line number at which it appears in the CAFD. This may require the decomposition of lines that contain multiple or complex statements. The preprocessing step also ensures that all inputs to operations in the CAFD are exposed as CAFD variables. For example, complex arithmetic expressions such as $d = a + b + c$ (allowable only when two adders can be chained) would be broken into $tmp = b + c$ and $d = a + tmp$. This can increase the number of variables in the CAFD in general, but from our experience the attendant overhead in code size and execution time is quite small. Some of the other translations performed in the preprocessing step are listed in Table I.

TABLE I
EXAMPLE PREPROCESSING CODE TRANSLATIONS

Original	Translated	Notes
A++;	A = A+1;	
A @ = B;	A = A @ B;	@: any binary operator
A=B+C; D=E+F	A=B+C; D=E+F;	

B. RTL Information Extraction

The RTL information extraction step correlates RTL components to CAFD functional elements, and establishes relationships between component inputs/outputs and CAFD variables.

For each state in a CAFD, we generate a *mapping table* to map its functional elements to RTL components. The table also records the type and bit-width of the RTL components, the names of inputs and output, and the RTL components to which these names are mapped.

An RTL implementation not only provides binding information, but also connectivity information. We need the synthesizer to record the connectivity information of each multiplexer, *i.e.*, which RTL components are connected to its data inputs¹. A *connectivity table* with such information is generated for each multiplexer that drives the input of a functionally-explicit RTL component such as a functional unit or register. Furthermore, we generate a *select-signal table* for each multiplexer that specifies which of its data inputs is selected in each control state. In states where the functionally-explicit component driven by the multiplexer is active, the select signal value can be determined by simply examining which multiplexer input needs to be routed to the component for it to perform the CAFD operation mapped to it. In states where the functionally-explicit component driven by the multiplexer is idle, this information can be deduced by analyzing the code of control logic that feeds the multiplexer select signals in the RTL implementation. Whenever the values cannot be decided statically, a random choice is made.

The above information is used by the virtual component instantiation and idle cycle analysis techniques described later in this section.

C. Virtual Component Instantiation and I/O mapping

A VC is instantiated for each functionally-explicit RTL component and each multiplexer to keep a record of previous and current input vectors. Power estimation is the only purpose for instantiating a VC. It has no effect on the circuit functionality. Fig. 4 shows an example implementation of a

¹For the sake of efficiency, we consider a multiplexer tree as an atomic n -to-1 multiplexer.

```

template <int type, int bitwidth>
class VC {
private:
    /* pointer to RTL power model library
     * power data */
    int *power_data;
private:
    int current_in1, current_in2;
    int previous_in1, previous_in2;
    /* pointers to the RTL components
     * that output to myself */
    VC *in1;
    VC *in2;
    /* output value */
    int current_out, previous_out;
public:
    double CalculatePower();
    /* To record the input VCs */
    void RecordInputVC(VC *I1, VC *I2);
    /* To record the I/O values */
    void RecordInput(int I1, int I2);
    void RecordOutput(int output);
    /* To set power_data to the proper row */
    void initialize(int type, int bitwidth);
}

```

Fig. 4. C++ class for a virtual component

VC class for a particular class of two-input and single-output RTL components. The power data for different RTL library elements are stored in different rows of a two-dimensional array. The example VC has a pointer, *power_data*, which is initialized to the appropriate row according to the RTL component's properties, such as the type and bit-width recorded in the mapping tables. Each VC uses a circular queue of depth two to keep track of the input and output values for the current and previous cycles.

For a CAFD code line containing a functional element, an update to the corresponding VC's I/O values is performed, by capturing the values of the appropriate CAFD variables with *RecordInput()* and *RecordOutput()*. For example, a part of the RTL-aware CAFD for the GCD circuit is shown in Fig. 5, wherein the VC updates for control state *ST_2* are shown in detail. Note that the I/O updates described above only affect components that are active in the current cycle. Optionally, each VC also contains pointers to the VCs that drive its inputs. For example, the VC corresponding to the subtractor in the GCD circuit contains pointers to the VCs corresponding to the two multiplexers that drive its inputs. As seen later, this is used to get the input values for idle cycles. The pointers can be updated through *RecordInputVC()*.

It is worth noting that the execution flow of a CAFD can be dynamic. First, a state followed by different states under different conditions introduces *inter-state dynamics*. Second, within a state-functional description, code may be conditionally executed as in the case of GCD *ST_3*, introducing *intra-state dynamics*. Inter-state dynamics are captured by placing power calculation code (*CalculatePower()*) just before all possible state-transition points. Intra-state dynamics are captured by inserting I/O updating code just before/after the statement of interest. The inserted code will be executed only when the corresponding statement is executed. Intra-state dynamics are also the reason that select-signal values of some

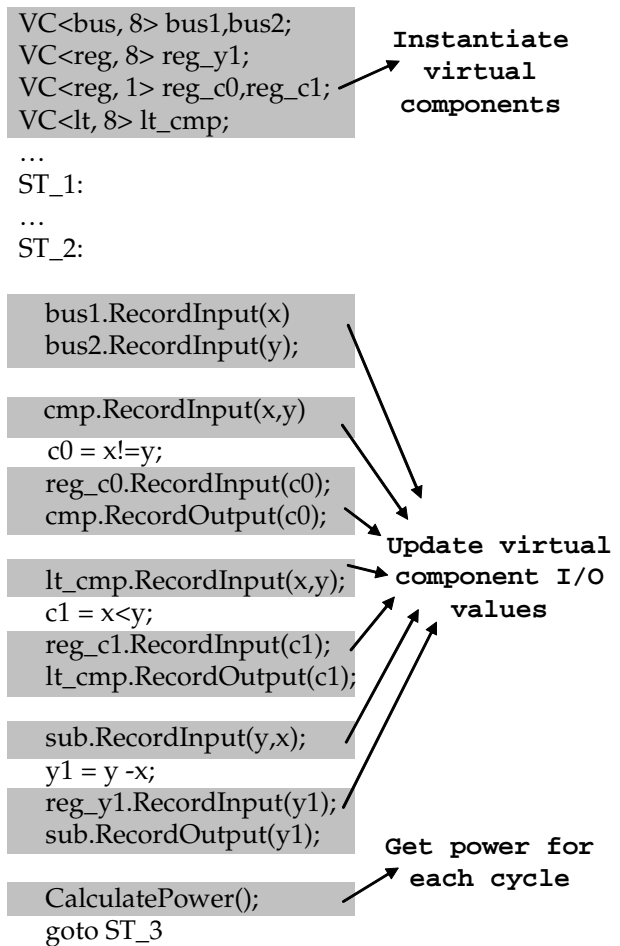


Fig. 5. A portion of the RTL-aware CAFD for the GCD example

multiplexers may not be statically determined for the select-signal table, introducing errors in power estimation.

D. Idle-cycle Handling

For any given control state in the CAFD, the I/O values of idle RTL components cannot be directly deduced from the CAFD or the mapping tables. In general, this is a difficult problem if the RTL circuit has an arbitrary structure. Fortunately, most high-level synthesis tools generate RTL circuits that are structured to have multiplexers at the inputs of functionally-explicit components. Furthermore, the inputs to these multiplexers come from the outputs of other functionally-explicit components. Given this property, idle-cycle inputs to a component can be inferred from the implementation style of the component's input multiplexers. For example, if an AND-OR based selector is used to implement the multiplexer, the multiplexer's output is set to zero in idle cycles. Alternatively, if a tristate-based multiplexer is used, the multiplexer's output is set to the same value as in the previous active cycle. For most other multiplexer implementations, one of the multiplexer's inputs is routed to its output.

When input multiplexers of a switch-case style are used, the input value is set to the output value of a fixed RTL

component for all idle cycles. Finally, if input multiplexers latch the selections in the previous active cycle like the retentive multiplexer in [15], an RTL component takes input values from the same ports that it did in the previous active cycle.

All the above situations can be handled by VCs, as they can record both the values of inputs and the pointers to the VCs for the RTL components connected to their inputs in the previous active cycle. The key is to be able to identify the style of the input multiplexers used during synthesis.

E. Sources of Error

Theoretically, our approach guarantees the same accuracy as RTL power estimation for functionally-explicit RTL components, which make up the circuit datapath. However, functionally-implicit components (multiplexers and control logic) impose a limit on the accuracy our approach can achieve.

Luckily, large industrial application-specific integrated circuits (ASICs) usually have relatively small controllers compared to their datapaths. For example, the combinational components of the controllers contribute 1%-3% to the total power in our benchmark circuits. We estimate the power consumption in the control logic by analyzing each controller state transition using a constant switching activity factor for the control inputs from the datapath. The resulting numbers are used to generate the power consumed by the control logic in each state. While similar to previous work on functional modeling of FSM power [14], this approach can lead to a small amount of estimation error.

Multiplexers are much more important in terms of power consumption. Therefore, VCs are instantiated for them. The connectivity and select-signal tables enable us to obtain the input values to a multiplexer in every state. Error is introduced only when a random choice is made in select-signal table generation due to intra-state dynamics, as discussed in Section III-B. As demonstrated through our experimental results, these sources of error do not have a significant impact.

IV. ADAPTIVE STATE-BASED SAMPLING

The basic approach detailed in the previous section updates the VCs and calculates power for *every component* in *every cycle*. The associated computation overhead can slow down the simulation manifold, depending on the implementation. In this work, we explore the applicability of sampling to CAFD power estimation, and develop a specific sampling technique, called adaptive state-based sampling, that is highly effective in our context.

In its simplest form, sampling, or temporal sampling, is implemented by performing power estimation only during a randomly chosen subset of the simulation cycles. The rationale is that the power statistics (*e.g.*, average power) computed during the sampled cycles is a reasonable approximation for the power consumed in the remaining cycles. Many different flavors of sampling have been explored for gate-level power estimation, and a few efforts have been made in the context of RTL power estimation as well. In [12], [16] and the works

that followed, the authors investigated how input traces for power simulation can be reduced to improve *average power* estimation efficiency. Unlike all the previous work, our work focuses on improving estimation speed while maintaining low cycle-by-cycle errors (low AACPE). In [27], the authors devote more computation power to more important RTL components in terms of power consumption and variation. With a similar philosophy, *node sampling* was studied in [5] to reduce RTL power estimation to gate-level power estimation with sampling. These spatial sampling techniques are complementary to our work, and can be readily used to alleviate the “every-component” problem, *i.e.*, by targeting only the important components.

In this paper, we propose solutions to the “every-cycle” problem, by targeting only the important cycles for expending computational effort for power estimation. Our technique works as follows. During CAFD simulation, for each simulation cycle, we use a sampling probability to determine whether or not detailed power estimation will be performed in the current cycle. The value of this probability is dependent on which control state of the CAFD is executed (hence the term state-based). Furthermore, the sampling probability is adaptively varied over time to tightly control the estimation error, as described later in this section (hence the term adaptive).

In cycles chosen for sampling, we invoke the power macromodels for each component, and aggregate the power consumed by all the components, as described in Section III. In order to produce power estimates for cycles that are not chosen for sampling, we maintain a small amount of power consumption history for each control state in the CAFD. For example, for a state *ST_1*, we maintain the power consumption calculated during the last *k* sampled cycles for which the CAFD was in state *ST_1*. We view this state-based history of power values as a time series for which we need to predict the next value. This is achieved using simple functions of the history values to estimate the power consumed in the current simulation cycle.

In contrast to temporal sampling approaches used at the gate level, our technique exploits an understanding of the CAFD structure, by performing independent sampling and maintaining a separate power history for each control state. This leads to high accuracy for cycle-by-cycle power estimates in addition to accurate average power estimates.

A. Rationale

The rationale for the proposed adaptive state-based sampling strategy is as follows:

- The power consumption characteristics of circuits are quite different when they are in different control states. Some control states exhibit a high variance in power consumption, while other states display a relatively predictable behavior.
- Some circuits display significantly time-varying power characteristics. Sampling techniques that ignore this may generate accurate average power estimates, but usually result in poor cycle-by-cycle estimates.

In order to illustrate the above observations, we consider an example design HDTV-1, which is an image filter module used

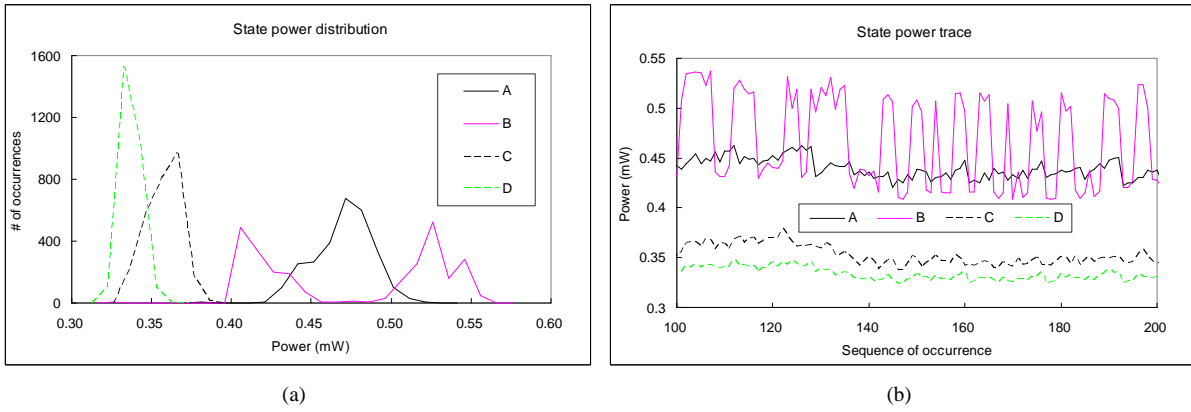


Fig. 6. Power characteristics for four states of the HDTV-1 design: (a) power distributions, and (b) power vs. time waveforms

in an SoC for high-definition television (HDTV) applications. The CAFD for the HDTV-1 design contains more than 20 control states, of which we have selected four representative states for our discussion, namely, A, B, C, and D.

Fig. 6(a) shows the power histograms for the HDTV-1 circuit when it is in each of the four states, A through D. This information was derived using a commercial RTL power estimation tool [31]. The X-axis in Fig. 6(a) indicates the power consumption in milliWatts (mW), while the Y-axis indicates the number of occurrences of that state with the given power consumption. We can see that the power distribution of different states can be quite different in terms of mean and standard deviation. The distributions for states A, C, and D are single-peaked, while state B’s power distribution is double-peaked.

Fig. 6(b) plots the power consumption for each state over time. The X-axis represents the occurrence number of that state, *i.e.*, the first time the state occurs during the simulation, the second time it occurs, and so on. Again, it is quite clear that different states have significantly differing power characteristics. In particular, state B displays a relatively large variation over time.

Fig. 7 shows a mean *vs.* standard deviation scatter plot that summarizes the state power statistics for four of our benchmark circuits. Note that only states with a statistically significant number of occurrences are presented. Each point in the figure represents a state (different point markers are used for different circuits). The (x,y) co-ordinates of a point represent the mean and standard deviation of the power consumed when the circuit is in the corresponding control state. The figure validates the observation that only a limited number of states display high standard deviation in power. Power for these states is “difficult to estimate,” and deserves a higher allocation of computational effort (this is achieved by giving the states a higher sampling frequency). Other states have a relatively low standard deviation, and can be handled with low sampling frequencies.

B. Proposed Sampling Technique

The above observations motivate us to consider sampling (calculating) a state’s power consumption in only some of its

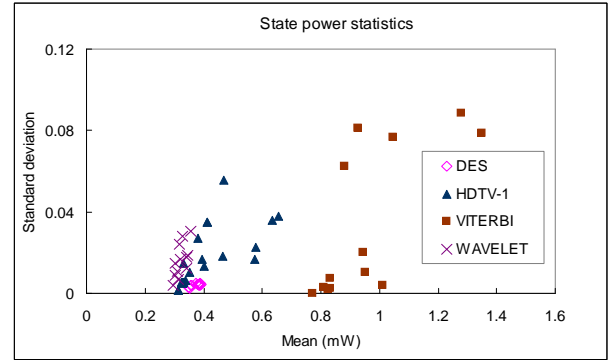


Fig. 7. Scatter plot of mean *vs.* standard deviation of power consumption in different states for four benchmarks

occurrences, and estimating it in others based on past samples. Two questions need to be answered for this purpose: when to sample, and how to estimate power using the history. We address these questions in the rest of this sub-section.

1) *Adaptive state-based sampling*: As we have seen before, different states have different power value localities and temporal power variations, which suggests that we devote more computing resources to states whose power varies a lot and to occurrences of a state in which power varies faster. In sampling techniques, the sampling probability is the “knob” that can be used to control the amount of computational effort allocated. Therefore, we propose a feedback-driven adaptive sampling scheme to determine a sampling probability for each state. In this scheme, all states start with the same sampling period. Whenever a state’s power is sampled, the sampled “real” power is compared with the “estimated” value (estimation will be addressed next). If the observed ACPE is larger than a *maximum error threshold*, the state’s sample period is decreased by one “step” unless the period has already reached the *minimum period*. Otherwise, if the ACPE is smaller than a *minimum error threshold*, the sample period is increased by one step unless it has already reached the *maximum period*.

Note that the minimum and maximum periods are used to control the adaptation so that it does not go too far. In all our experiments, they are set as 1 and 30 occurrences, respectively. In theory, if the maximum period is too large, a

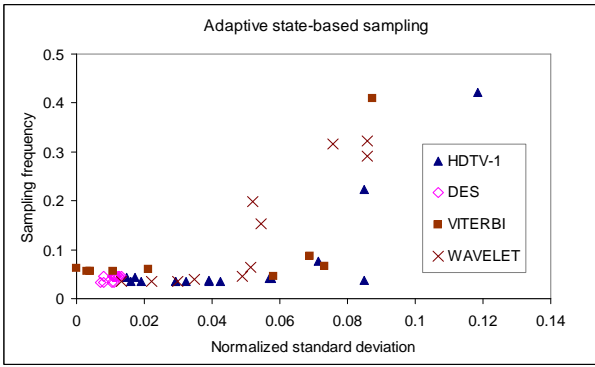


Fig. 8. Effect of adaptive sampling: Sampling frequency allocated to the different states by our technique

state’s sampling period may become so large that adaptation is irresponsive to errors. However, our experiments showed that accuracy degrades only slightly even when the maximum period is relaxed to infinity.

The adapting step controls the adaptation granularity, and is set as two occurrences in all our experiments. If the step size is too large, while the difference between the minimum and maximum error thresholds is too small, the adaptive sampling scheme could oscillate, *i.e.*, the sample period for a state may be repeatedly increased and decreased. For a step of two occurrences, and when the ratio of the minimum error threshold to the maximum error threshold is less than 0.8, we found that oscillation never occurred in practice. Since oscillation does not introduce much overhead, even when it happens, adaptive sampling still achieves better speed-accuracy tradeoffs than fixed-period sampling.

The speed-accuracy tradeoff can be controlled in our adaptive sampling technique by changing the values of the various parameters described above. A shorter step and tighter error thresholds result in higher accuracy at the cost of increased computational effort.

The net effect of the adaptive state-based sampling technique is to optimize the allocation of sampling probabilities to different control states such that states with a higher standard deviation or higher time-variance of power will be sampled more frequently. In order to illustrate this, we simulated the proposed technique with the power traces generated by the CYBER RTL power estimation tool [31] for four of the benchmark circuits. During power estimation, we computed the overall sampling frequency for each state in each benchmark. A scatter plot of sampling frequency *vs.* normalized standard deviation of power for each state is presented in Fig. 8. Normalized standard deviation is standard deviation divided by the mean. It is used to reflect the percentage power variance since ACPE is used for error control. As expected, control states with higher normalized standard deviation are allocated higher sampling frequencies.

We also plot the variation of the sampling period over time, for the four states, A through D, in the HDTV-1 benchmark in Fig. 9. Note that those for states A and C are shifted upward by 40 and 20 occurrences, respectively, for better visibility. From Fig. 6, we can see that state B has a relatively

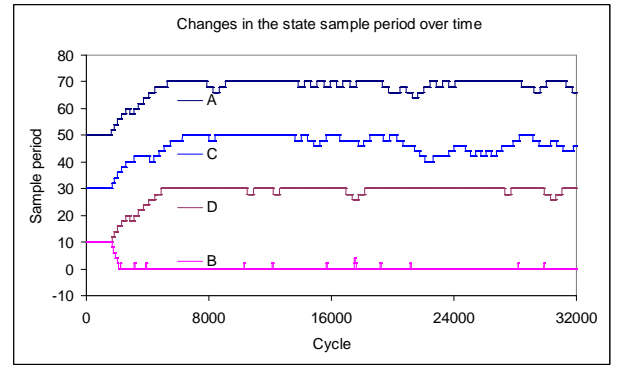


Fig. 9. Variation in the sampling period over time, for four different states of the HDTV-1 design

high standard deviation and exhibits higher power variation over time. As a result, the adaptive state-based sampling technique decreases the sampling period for state B (*i.e.*, increases the sampling frequency), in this case to the minimum value. On the other hand, the sampling frequencies for states A, C, and D are initially increased to the maximum value, but subsequently adapted when errors above the maximum threshold are observed.

2) *History-based estimation policy*: Next, we address the how-to-estimate question. Unlike the classical time series prediction problem, the history we have for the power consumption of a state is quite sporadic since we only have a sampled, instead of complete, history. Moreover, since power estimation has to be carried out in every cycle, it has to be very efficient. We experimented with several choices. The simple estimation can be based on the *mean* of past samples. If we assume the state power has a normal distribution and different occurrences of the same state behave independently with each other, they can be viewed as a stationary Gaussian time series [6], for which the minimal mean square error is achieved when the mean of the past values is used as the prediction for the next occurrence. However, we observed that different occurrences of the same state are actually slightly related to each other and the autocorrelation drops rapidly as the distance (lag) between samples increases. Such vanishing dependence makes the mean prediction not as good as a mean of a limited history, which is in turn worse than the weighted mean (W_{mean}) of a limited history with smaller weights for older samples. Our experiments show that such a weighted mean slightly outperforms the mean and significantly outperforms extrapolation-based estimation. The tradeoff curves (number of samples *vs.* AACPE) for HDTV-1 are shown in Fig. 10(a). Based on our experiments, the weighted mean approach is adopted in our implementation.

Another concern is the history size, *i.e.*, the number of past samples, used for estimation. Fig. 10(b) plots the speed-accuracy tradeoff curves for different history sizes for HDTV-1 using the weighted-mean approach. It can be seen that increasing the history size from four to eight or 16 samples does not yield much accuracy benefit. Hence, unless otherwise indicated, four past samples are used in all our experiments. The power consumption, $P_s(n)$, for the n th occurrence of state

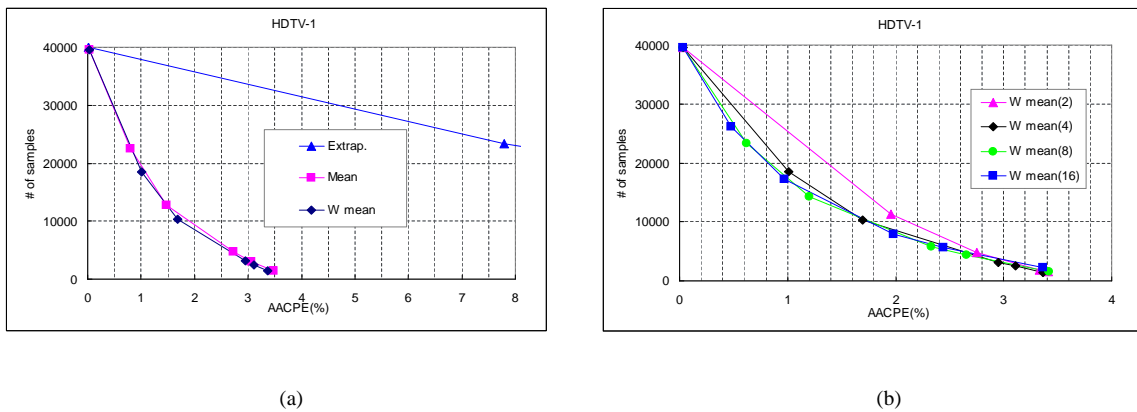


Fig. 10. Comparison of the effectiveness of different estimation policies: (a) estimation function, and (b) history size used in estimation

s is estimated as

$$P_s(n) = [4 \cdot P_s(m_1^S) + 3 \cdot P_s(m_2^S) + 2 \cdot P_s(m_3^S) + P_s(m_4^S)] \cdot 0.1$$

where m_1^S , m_2^S , m_3^S , and m_4^S are the most recent four occurrences of s (m_1^S being the most recent, then m_2^S , and so on), for which power is sampled instead of estimated. Such an estimation is much simpler than using RTL power models, and results in substantial speedup as shown in the next section.

It is worth mentioning that the average power errors for all the tradeoff points shown in Fig. 10(a) are within 1%, which shows how much more challenging cycle-accurate power estimation is than average power estimation.

C. Intra-state Dynamics and Inter-state Dynamics

As has been illustrated by states A, C, and D of HDTV-1, many states of large circuits have single-peaked or normal-distribution-like power distributions, which can be explained intuitively by the central limit theorem [22]. However, some states, like state B, do exhibit multiple-peaked power distributions, which keep their state sampling frequencies high and prevent our adaptive state-based sampling approach from accelerating power estimation. A close examination reveals that many such multiple-peaked power distributions can be attributed to intra-state dynamics as part of the circuit is conditionally activated when there is conditional code in the state-functional description.

Intra-state dynamics partition the state behavior into subspaces so that the central-limit theorem is no longer relevant and each subspace is likely to contribute a peak to the power distribution. However, intra-state dynamics can be reduced or even eliminated for functional simulation/power estimation without changing the CAFD behavior by adding more states and corresponding state-functional description. For example, intra-state dynamics in ST_3 of the GCD CAFD (see Fig. 1(b)) can be eliminated by adding two states, as shown in Fig. 11. Since intra-state dynamics introduce power estimation error as described in Section III, eliminating it will also improve power estimation accuracy. It is obvious that intra-state dynamics are reduced only at the cost of more inter-state dynamics.

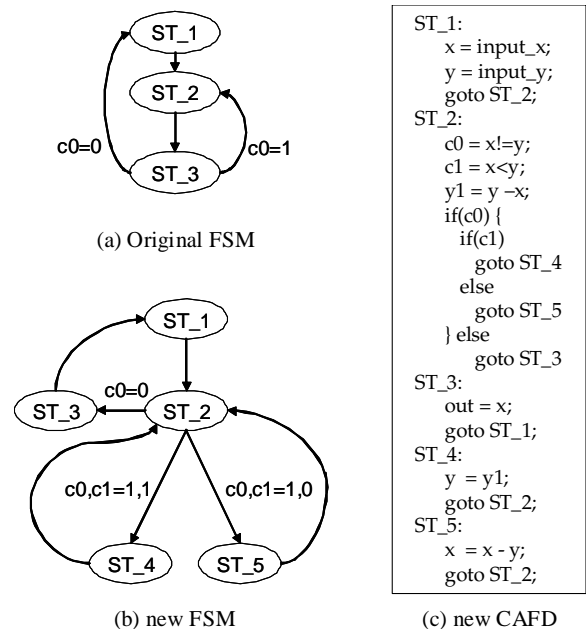


Fig. 11. GCD: eliminating intra-state dynamics

V. EXPERIMENTAL RESULTS

In this section, we first describe how the proposed CAFD power estimation techniques were integrated in the context of a commercial C-based design flow. We then present the results of applying the techniques to a number of large industrial designs.

A. Implementation

We implemented the proposed RTL-aware functional power estimation and adaptive state-based sampling approaches in the context of the CYBER C-based design flow [31]. For any input functional description and resource constraints, CYBER performs high-level synthesis and generates an optimized RTL description in VHDL and the corresponding CAFD in C or SystemC. The CYBER tool flow also provides an RTL power estimation tool that uses pre-characterized power macro-models (also described as simulate-able VHDL entities) for various RTL components. It generates a power-estimation

enhanced VHDL, which can be simulated with ModelSim [19] to produce a cycle-accurate power report.

1) *RTL-awareness*: CYBER tags the output RTL VHDL description and C-based CAFD with the corresponding code line numbers of the input functional description for the purpose of debugging. We were able to generate most of the RTL information discussed in Section III-B by matching the tags in both the RTL VHDL description and the C-based CAFD. We first preprocessed the functional description in a fashion very similar to that described in Section III-A so that tag matching is facilitated. Then we used CYBER to synthesize the preprocessed functional description into an RTL description in VHDL and the corresponding CAFD in C, subject to resource constraints and synthesis options. We implemented a tag matcher that generates the mapping tables and connectivity tables by correlating functionally-explicit elements with RTL components through matching of the corresponding tags. The multiplexer implementation information was deduced from the synthesis options of CYBER for idle cycle handling.

2) *VC instantiations*: We implemented a script that converts the RTL VHDL power macro-models into C code, which consists of more than 25K lines. It is worth noting that our approach is independent of how the RTL power macro-models are built. We implemented a library of VC classes, as outlined in Section III-C. The VC library consists of about 3.4K lines of C++ code (note that the RTL power library has more than 58K lines of code). Another script instantiates VCs in the CAFD based on the power library implementation, and generates the I/O mapping code. During simulation, previous and current input values recorded by VCs are input to the power models, which can calculate the power consumption in each cycle.

3) *Adaptive state-based sampling*: We implemented the proposed adaptive state-based sampling approach as a stand-alone C library. Instead of calculating power every cycle or on each state occurrence, the RTL-aware CAFD calls routines in the library for adaptive state-based sampling. The library determines whether the circuit power consumption for an occurrence of a state is calculated using VCs and power models, or is estimated using the power consumption history of the state. The policy parameters associated with the adaptive mechanism can be set by command line options. For the experiments, the settings mentioned in Section IV were used.

4) *Usage*: The VC-instantiated CAFD with sampling-based power estimation can be finally compiled into an executable using *gcc*. A large number of command line options (such as whether adaptive sampling is used, sampling parameters, number of cycles to simulate, how power report is dumped, etc.) are provided for flexibility. The power profile for different types of RTL components or even individual RTL component instances can be generated.

It is worth noting that synthesis by CYBER and our post-processing step take little time, finishing in tens of seconds, while RTL power estimation takes minutes to hours, and even days for the largest benchmarks.

B. Benchmarks

We performed power estimation on a number of large industrial designs using our prototype implementation. Simulations

TABLE II
BENCHMARK INFORMATION

Circuit	Number of code lines ($\times 10^3$)				Gate count
	CAFD in C		RTL in VHDL		
	Orig.	Enh.	Orig.	Enh.	
DES	5.9	6.3	3.6	9.9	5,845
HDTV-1	7.5	9.3	4.5	10.0	12,118
JPEG	66.7	79.2	41.8	106.1	1,187,696
MPEG4-IDCT	5.6	6.2	3.3	6.3	11,227
MPEG4-ISPQ	13.5	15.0	6.9	17.9	49,262
SORT	8.3	10.2	1.9	8.8	4,574
VITERBI	8.7	12.6	11.3	18.0	47,655
WAVELET	1.1	1.6	1.5	2.0	257,918

were performed on a SUN Fire 280R server with two 900-MHz Ultra-Sparc processors and 4GB RAM. Table II reports statistics for our benchmark designs, which correspond to complete ASICs, as well as components of industrial SoCs. DES, JPEG, SORT, VITERBI, and WAVELET are designs that implement the Digital Encryption Standard encryption, JPEG decoding, bubble-sort algorithm, Viterbi decoding, and a Wavelet-based image filter, respectively. HDTV-1 is a filter module in an industrial SoC design for HDTV decoding, while MPEG4-IDCT and MPEG4-ISPQ are two modules in an industrial SoC design for MPEG4 decoding. Columns 2 and 3 indicate the number of code lines in the original (Orig.) and power-estimation enhanced (Enh.) CAFD. Columns 4 and 5 report the corresponding numbers for the RTL VHDL descriptions. Column 6 reports the gate counts for the technology-mapped net-lists (mapped to a commercial 0.18μ technology [20]) obtained after logic synthesis using Synopsys Design Compiler [9].

C. Results

For each benchmark, we first employed the RTL power estimation tool to obtain the reference cycle-accurate power report. We then compared the corresponding numbers generated from the CAFD power estimation with the reference report to obtain the AACPE and the error of accumulative (average) power estimation. All benchmarks were simulated for 40K cycles, except SORT (2600 cycles) and VITERBI (22K cycles).

Table III summarizes the results for the basic CAFD power estimation approach in which sampling is not used. Note that “Speedup” (Column 4) refers to the speedup of our approach over RTL power estimation, while “Slowdown” (Column 6) compares the performance of our approach with pure C-based cycle-accurate functional simulation without power estimation, which is actually an upper bound for the speed of cycle-accurate functional power estimation. The results show that significant speedup is achieved with little sacrifice in cycle-by-cycle power accuracy.

We have already seen the advantage of the adaptive state-based sampling approach in Section IV. Table IV summarizes the results when it is used along with the basic CAFD power estimation approach for all the benchmarks. Column 4 reports

TABLE III
EFFICIENCY AND ACCURACY FOR THE BASIC APPROACH

Circuit	Error (%)		Speedup (X)	Slowdown (X)
	Avg.Error	AACPE		
DES	2.1	2.2	73	1.3
HDTV-1	1.5	3.9	150	7.5
JPEG	2.8	6.5	331	11.3
MPEG4-IDCT	3.1	4.7	214	6.1
MPEG4-ISPQ	1.2	2.2	177	5.2
SORT	1.6	5.5	148	3.0
VITERBI	1.2	5.9	123	7.5
WAVELET	2.3	3.8	129	3.6

TABLE IV
EFFICIENCY AND ACCURACY WITH ADAPTIVE STATE-BASED SAMPLING

Circuit	Error (%)		Speedup (X)	Slowdown (X)
	Avg.Error	AACPE		
DES	2.1	2.2	83	1.1
HDTV-1	1.7	4.0	356	3.2
JPEG	2.7	6.6	1,143	3.3
MPEG4-IDCT	3.1	5.1	412	3.2
MPEG4-ISPQ	1.5	2.4	438	2.1
SORT	1.7	5.4	266	1.7
VITERBI	1.4	6.5	305	3.0
WAVELET	2.4	5.1	223	2.1

the speedup with respect to RTL power estimation. Column 5 shows that the power simulation speed with adaptive state-based sampling is, at worst, only about three times slower than the bound set by cycle-accurate functional simulation without power estimation. Such a slowdown is mainly caused by virtual component I/O updates in every cycle and power calculations in the sampled cycles.

To offer a closer look at the ACPEs, Fig. 12 plots the ACPE distributions for four of the benchmarks. It shows that more than 50% of the ACPEs are within 5% and more than 80% are within 10%. The speedups reported in Tables III and IV include constant simulation overheads such as binary loading and program initiation. We performed an additional experiment in order to study the variation of execution time with the length of simulation (Fig. 13) for the HDTV-1 benchmark. The results show that C-based power estimation is asymptotically more than 180 times faster than RTL VHDL power estimation in this case. The use of adaptive sampling further doubles this speedup.

Finally, Fig. 14(a) shows the cycle-by-cycle power trace generated by the CYBER RTL power estimation tool and Fig. 14(b) shows those generated by the adaptive state-based sampling approach and other two alternative sampling approaches, cycle-based and random cycle-based. The *cycle-based* approach obtains cycle power samples periodically (every 10 cycles). The *random* cycle-based method samples the cycle power with a probability of 0.1. Neither of these two approaches is state-aware. The power trace generated by adaptive state-based sampling matches the trace generated by

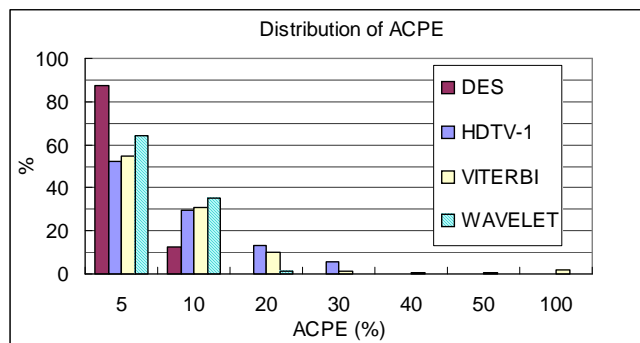


Fig. 12. ACPE distribution

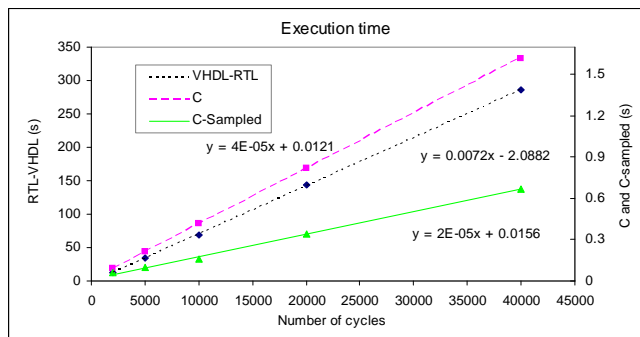


Fig. 13. Execution time for RTL-VHDL and our C-based power estimation

the RTL power estimation tool much better than the other two. Moreover, the adaptive-state sampling approach is two times faster. This clearly demonstrates the advantage of state-awareness when utilizing sampling techniques. It is also worth noting that the average power errors for all three traces are within 2%. This again shows that cycle-accurate power estimation is much more challenging than average power estimation.

VI. CONCLUSIONS

In this paper, we proposed a fast and accurate methodology for obtaining cycle-accurate power estimation for functional descriptions of hardware. Our methodology leveraged the correlation that exists between a CAFD and a lower-level (RTL) structural implementation, whose components have been pre-characterized for their power consumption behavior. We provided efficient techniques for augmenting functional descriptions with limited RTL-awareness as well as the code needed for power estimation. We also proposed adaptive state-based sampling to further improve the efficiency of cycle-accurate power estimation. We validated the proposed framework in the context of an industrial C-based design flow and evaluated its performance with a number of industrial benchmark designs. The results indicate that cycle-accurate power reports can be generated with very high efficiency (two-to-three orders of magnitude speedup compared to RTL power estimation). In effect, the proposed power estimation approach has a speed close to functional simulation, with accuracy close to RTL power estimation. Although we proposed and

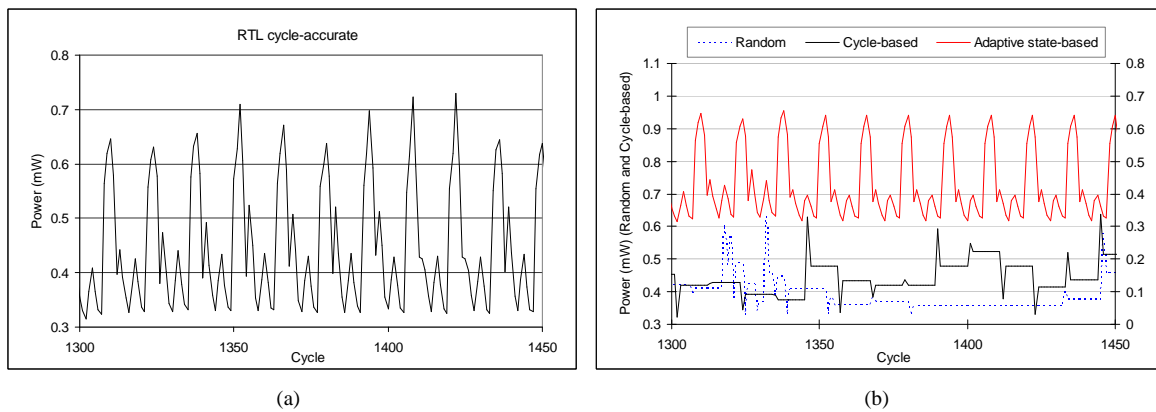


Fig. 14. Cycle-by-cycle power traces generated by (a) RTL power estimation, and (b) RTL-aware functional power estimation with different sampling approaches

implemented the back-annotated power estimation and adaptive state-based sampling methodologies specifically for cycle-accurate functional power estimation, the same philosophy can be applied to other levels of design abstraction as well.

REFERENCES

- [1] B. Bailey and D. Gajski, "RTL semantics and methodology," in *Proc. Int. Symp. System Synthesis*, Oct. 2001, pp. 69–74.
- [2] A. Bogliolo, L. Benini, and G. De Micheli, "Adaptive least mean square behavioral power modeling," in *Proc. Design Automation & Test in Europe Conf.*, Mar. 1997, pp. 404–410.
- [3] —, "Characterization-free behavioral power modeling," in *Proc. Design Automation & Test in Europe Conf.*, Feb. 1998, pp. 767–773.
- [4] A. Bogliolo, I. Colonescu, E. Macii, and M. Poncino, "An RTL power estimation tool with on-line model building capabilities," in *Proc. Int. Wkshp. Power & Timing Modeling, Optimization & Simulation*, Sept. 2001, pp. 391–396.
- [5] A. Bogliolo and L. Benini, "Node sampling: A robust RTL power modeling approach," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1998, pp. 461–467.
- [6] P. Brockwell and R. A. Davis, *Introduction to Time Series and Forecasting*. New York: Springer-Verlag, 1996.
- [7] A. Chandrakasan and R. Brodersen, *Low-Power CMOS Design*. Wiley-IEEE Computer Society Press, Mar. 2001.
- [8] Z. Chen and K. Roy, "A power macromodeling technique based on power sensitivity," in *Proc. Design Automation Conf.*, June 1998, pp. 678–683.
- [9] Design Compiler, <http://www.synopsys.com>.
- [10] F. Ferrandi, F. Fummi, E. Macii, M. Poncino, and D. Sciuto, "Power estimation of behavioral descriptions," in *Proc. Design Automation & Test in Europe Conf.*, Feb. 1998, pp. 762–766.
- [11] S. Gupta and F. N. Najm, "Energy and peak-current per-cycle estimation at RTL," *IEEE Trans. VLSI Systems*, vol. 11, no. 4, pp. 525–537, Aug. 2003.
- [12] C.-T. Hsieh, Q. Wu, C.-S. Ding, and M. Pedram, "Statistical sampling and regression analysis for RT-level power evaluation," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1996, pp. 583–588.
- [13] L. Kruse, E. Schmidt, G. Jochens, A. Stammermann, A. Schulz, E. Macii, and W. Nebel, "Estimation of lower and upper bounds on the power consumption from scheduled data flow graphs," *IEEE Trans. VLSI Systems*, vol. 9, no. 1, pp. 3–14, Feb. 2001.
- [14] P. Landman, "High-level power estimation," in *Proc. Int. Symp. Low Power Electronics & Design*, Aug. 1996, pp. 29–35.
- [15] J. Luo, L. Zhong, Y. Fei, and N. K. Jha, "Register binding based RTL power management for control-flow intensive designs," *IEEE Trans. Computer-Aided Design*, no. 8, pp. 1175–1183, Aug. 2004.
- [16] R. Marculescu, D. Marculescu, and M. Pedram, "Adaptive models for input data compaction for power simulator," in *Proc. Asia & South Pacific Design Automation Conf.*, Jan. 1997, pp. 391–396.
- [17] R. Mehra and J. Rabaey, "Behavioral level power estimation and exploration," in *Proc. Int. Wkshp. Low Power Design*, Apr. 1994, pp. 197–202.
- [18] H. Mehta, R. M. Owens, and M. J. Irwin, "Energy characterization based on clustering," in *Proc. Design Automation Conf.*, June 1996, pp. 702–707.
- [19] ModelSim 5.7, <http://www.model.com>.
- [20] NEC cell-based ASIC CB-11. <http://www.necel.com/ASIC/>, 2000.
- [21] M. Nemani and F. Najm, "Towards a high-level power estimation capability," *IEEE Trans. Computer-Aided Design*, no. 6, pp. 588–598, June 1996.
- [22] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*. New York: McGraw-Hill, 1984.
- [23] N. R. Potlapally, A. Raghunathan, G. Lakshminarayana, M. Hsiao, and S. T. Chakradhar, "Accurate power macro-modeling techniques for complex RTL components," in *Proc. Int. Conf. VLSI Design*, Jan. 2001, pp. 235–241.
- [24] J. Rabaey and M. Pedram, *Low Power Design Methodologies*. Kluwer Academic Publishers, June 1996.
- [25] A. Raghunathan, S. Dey, and N. K. Jha, "High-level macro-modeling and estimation techniques for switching activity and power consumption," *IEEE Trans. VLSI Systems*, no. 4, pp. 538–557, Aug. 2003.
- [26] A. Raghunathan, N. K. Jha, and S. Dey, *High-Level Power Analysis and Optimization*. Norwell, MA: Kluwer Academic Publishers, 1998.
- [27] S. Ravi, A. Raghunathan, and S. Chakradhar, "Efficient RTL power estimation for large designs," in *Proc. Int. Conf. VLSI Design*, Jan. 2003.
- [28] SpecC, <http://www.specc.org>.
- [29] SystemC, <http://www.systemc.org>.
- [30] SystemVerilog, <http://www.systemverilog.org>.
- [31] K. Wakabayashi and T. Okamoto, "C-based SoC design flow and EDA tools: An ASIC and system vendor perspective," *IEEE Trans. Computer-Aided Design*, vol. 19, no. 12, pp. 1507–1522, Dec. 2000.
- [32] Q. Wu, Q. Qiu, M. Pedram, and C.-S. Ding, "Cycle-accurate macro-models for RT-level power analysis," *IEEE Trans. VLSI Systems*, no. 4, pp. 520–528, Dec. 1998.