Strober: Fast and Accurate Sample-Based Energy Simulation for Arbitrary RTL

by Donggyu Kim

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Krste Asanović
Research Co-Advisor

(Date)

* * * * * * *

Professor Jonathan Bachrach
Research Co-Advisor

(Date)

Contents

List of Figures

List of Tables

Abstract

This thesis presents a sample-based energy simulation methodology that enables fast and accurate estimations of performance and average power for arbitrary RTL designs. Our approach uses an FPGA to simultaneously simulate the performance of an RTL design and to collect samples containing exact RTL state snapshots. Each snapshot is then replayed in gate-level simulation, resulting in a workload-specific average power estimate with confidence intervals. For arbitrary RTL and workloads, our methodology guarantees a minimum of four-orders-of-magnitude speedup over commercial CAD gate-level simulation tools and gives average energy estimates guaranteed to be within 5% of the true average energy with 99% confidence. We believe our open-source sample-based energy simulation tool Strober can not only rapidly provide ground truth for more abstract power models, but can enable productive design-space exploration early in the RTL design process.

Chapter 1

Introduction

Energy efficiency has become the primary design metric for both low-power portable computers and high-performance servers. As technology scaling slows down, computer architects must use architectural innovation rather than semiconductor process improvement to improve energy efficiency. This trend necessitates accurate and fast energy evaluation of various long-running applications on novel designs for architectural design-space exploration.

The most accurate way to evaluate energy efficiency is by running applications on a silicon prototype [1, 2, 3, 4, 5] with power consumption measured directly. Prototyping is accurate and can run large workloads rapidly, but each prototyping cycle is expensive and has a long latency, prohibiting extensive design-space exploration.

Computer architects instead mostly rely on analytic power models calibrated against representative RTL designs [6, 7, 8, 9, 10]. These must be driven by activities from micro-architectural simulation [11, 12, 13]. This approach helps designers gain some intuition in early design phases, but is limited to microarchitectures resembling those for which the abstract model was built, and requires long simulation times to gather microarchitectural activities. As power model validation depends on the existence of representative RTL, constructing abstract power models is more difficult for non-traditional architectures such as application-specific accelerators.

When complete RTL designs are available, they can be used to evaluate not only energy efficiency, but also cycle time and area using commercial CAD tools [14, 5, 15]. Although existing commercial CAD tools provide extremely accurate performance and power estimates from detailed gate-level simulation, the simulation runtime of complex designs is painfully slow, preventing large architecture studies of many hardware configurations.

This paper describes a sample-based RTL energy-modeling methodology, which enables fast and accurate energy evaluation of long-running applications. First, a design's performance is evaluated using full-system RTL simulation, during which a set of replayable RTL snapshots is captured randomly over the course of a program's execution. Next, the design's average power is estimated by replaying the samples on a gate-level power simulator, which also provides the confidence interval for the average power estimate.

We also present the open-source Strober framework, an example implementation of sample-based energy simulation. Strober is built upon Chisel [16], which supports advanced hardware designs using highly parameterized generators. Strober automatically generates an FPGA-accelerated FAME1 simulator [17] from any Chisel RTL design, to provide rapid performance modeling. The FAME1 simulator is enhanced with the ability to capture a full replayable RTL snapshot at any sample point, which can then be replayed on a commercial gate-level simulator to obtain power numbers. We evaluate the Strober framework using both an in-order processor [18] and an out-of-order processor [19].

The main contributions of this paper are as follows:

- *General and Easy-to-Use Framework:* Strober automatically generates FPGA-accelerated FAME1 simulations from any RTL design including the ability to snapshot simulation state for replay on gate-level simulation, thus minimizing designers' manual effort. We present results using RTL designs of in-order and out-of-order processors, but note that the approach applies to any Chisel RTL including application-specific accelerators.

- *Accurate Estimation:* Performance measurement is truly cycle-accurate, since it is based on the RTL design modeled using a token-based timing simulation. For average power, we can achieve less than 5% error with 99.9% confidence against commercial CAD tools. This indicates Strober can be a framework to provide ground truth for other models.

- *Fast Simulation:* We achieve more than two orders of magnitude speedup over existing microarchitectural simulators and four orders of magnitude speedup over commercial Verilog simulators. This implies Strober can support large design-space exploration using long-running applications on complex hardware designs.

1.1 Previous Publication, Collaboration, and Funding

Most of the content and figures in this thesis are adapted from a previous publication, “Strober: Fast and Accurate Sample-Based Energy Simulation for Arbitrary RTL” [20]. The previous publication and this thesis is the results of a collaborative group project. Although the Strober framework is mainly implemented by myself, other people have made direct contribution to the ideas included in this thesis. Adam Izraelevitz contributed to the ideas presented in Chapter 3 including reservoir sampling for random sampling on unknown length of simulations. Christopher Celio is the lead architect of BOOM, which was crucial to show the ideas of this thesis. He also contributed to the evaluation methodologies presented in Chapter 5 and Chapter 6. Hokeun Kim, my project parter of CS252, contributed to implementing DRAM activity counters for DRAM power estimates. Brian Zimmer suggests Synopsys Formality® to resolve name mangling from commercial CAD tools. Yunsup Lee, a leader in group meetings, shed light on how to handle register retiming in the Strober framework. Last but not least, Krste Asanović and Jonathan Bachrach, my research co-advisors, were integral in all aspects of the project.

This project is partially funded by DARPA Award Number HR0011-12-2-0016; the Center for Future Architecture Re- search, a member of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA; and ASPIRE Lab industrial sponsors and affiliates Intel, Google, HPE, Huawei, LGE, Nokia, NVIDIA, Oracle, and Samsung. I am supported in part by the Kwanjeong Educational Foundation.

Chapter 2

Related Work

Analytical power modeling [6, 7, 8, 9, 10] combined with microarchitectural software simulators [11, 12, 13] is widely-used for computer architecture research. This method enables early architecture-level design-space exploration, helping designers gain high-level intuitions before RTL implementation. However, microarchitectural software simulators execute far more slowly than real systems, requiring application runs to be subset. Moreover, the power models should be strictly validated against real systems or detailed gate-level simulations, which is difficult when exploring new non-traditional designs. We suggest sample-based energy simulation as a way of obtaining accurate ground truth to train abstract power models rapidly.

Power modeling based on performance-monitoring counters is also popular for power estimation [21, 22, 23, 24, 25, 26]. This method provides a quick power estimate by profiling full execution of applications. However, its application is limited to existing physical systems since standard power simulators are extremely slow. We believe the Strober framework enables the system designer to correlate power models and performance metrics of novel hardware designs by accelerating both performance and power simulations.

There are also significant efforts to validate power models [27, 28, 29, 30]. Shafi et al. [27] validate an event-driven power model against the IBM PowerPC 405GP processor. Mesa-Martinez et al. [28] validate power and thermal models by measuring the temperature of real machines. The authors measure temperature using an infrared camera and translate temperature to power using a genetic algorithm. Xi et al. [29] validate McPAT against the IBM POWER7 processor and illustrate how inaccuracies can arise without careful tuning and validation. However, these methodologies can only be applied using existing machines or proprietary data. Jacobson et al. [30] suggest a power model from systematically selected signals and validate it against RTL simulation. However, the approach relies on designer annotations and microbenchmarks exploiting familiarity with a particular family of processor architectures. In contrast, Strober can be used for validation of novel hardware designs and long-running real world applications.

There are a number of significant attempts to accelerate power estimation using an FPGA [31, 32, 33, 34, 35]. Sunwoo et al. [31] generate power models from manually specified signals, which requires designers' intuition. This technique also requires additional manual efforts to instrument existing FPGA simulators with power models. Bhattacharjee, Contreras, & Martonosi [32] also manually implement event counters in FPGA emulators to speed up event-driven power estimation. Coburn, Ravi, & Raghunathan [33] implement detailed power models directly on the FPGA, which suffers from large FPGA resource overhead. Ghodrat et al. [34] extend Coburn et al. by employing a software/FPGA co-emulation approach to reduce FPGA resource overhead, but introduces communication overhead between the software and FPGA, which can bottleneck emulation performance without careful partitioning. Atienza et al. [35] implement a special module to monitor selected signal activities on FPGA.

Our Strober framework differs in that the hardware design is automatically instrumented to generate samples instead of manually implementing power models on an FPGA, while still minimizing FPGA resource overhead.

Sampling procedures have been applied to speed up existing processor simulation frameworks; many such procedures, based on the foundational work of SMARTS [36], alternate at fixed intervals between detailed simulation (including a non-recorded warming stage) and fast functional simulation [13, 37, 38, 39]. This simulation methodology makes the following assumptions: (1) length of execution is known, (2) no aliasing along the fixed interval, (3) state warming terminates with an accurate state. While acceptable for simulating known architectures and known benchmarks, these assumptions are invalid when estimating power for arbitrary RTL running arbitrary code. Our proposed sample-based methodology avoids making these assumptions by employing reservoir sampling and cycle-accurate performance simulators.

There have been significant efforts to develop FPGA performance simulators [40, 41, 17, 42, 43, 44]. Protoflex [40] implements a multi-core functional simulator on the FPGA. FAST [41] is a hybrid approach simulating a function model in software and a timing model on the FPGA. Tan et al. [17] describe different FAME levels. FAME0 simulators directly emulate the RTL design on the FPGA. FAME1 simulators are decoupled from the host memory simulation to match the target DRAM timing models. FAME7 simulators implement abstract models and simulation multi-threading on top of FAME1.

RAMP Gold [17] and HASim [43] are examples of FAME7 simulators. The simulators above are orders of magnitude faster than software simulators, but they require significant simulator development efforts. In contrast, our approach automatically generates FAME1 simulators directly from RTL designs to accurately model the target design's timing behavior.

Chapter 3

Sample-based Energy Simulation

In this chapter, we present our sample-based energy simulation methodology using RTL designs for fast and accurate energy estimation. First, we present a brief theoretical background of statistical sampling in Section 3.1 with parameters in Table 3.1. Next, we describe how statistical sampling is applied to RTL energy simulation in Section 3.2.

3.1 Statistical Sampling

A population P of size N is the set of all elements (e_1, e_2, ... e_N) which could be selected in an experiment. Each element e_i has a corresponding measurable quantity, X_i. A population's parameters such as its mean, $\overline{X}$, and its variance, σ^2, can be exactly calculated if all elements within the population are measured.

$$\overline{X} = \frac{\sum_{i=1}^{N} X_i}{N} \tag{3.1}$$

$$\sigma^2 = \frac{\sum_{i=1}^{N} (X_i - \overline{X})}{N} \tag{3.2}$$

Unfortunately, evaluating every element in P is usually infeasible due to any number of resource constraints. Instead, a subset of the population, a sample, is selected according to a sampling strategy, and is used to estimate some parameters of the original population.

While there are many sampling strategies, the most statistically robust strategy is random sampling without replacement, where every e_i in P has an

Population		Sample	
size	N	*size*	n
mean	$\overline{X}$	*mean*	$\bar{x}$
variance	σ^2	*variance*	s_x^2
		sampling mean	$\overline{X}$
		sampling variance	$Var(\bar{x})$
		confidence level	$(1-\alpha)$
		confidence interval	$\bar{x} \pm z_{1-(\alpha/2)}\sqrt{Var(\bar{x})}$

Table 3.1: Statistical Parameters

equal probability of being selected in a sample. For simplicity and clarity, the following assumes this specific sampling strategy.

To estimate population parameters, every element in a sample of size n is measured (x_i), and the sample mean $\bar{x}$ and sample variance s_x^2 are calculated. These sample values are used to estimate the corresponding true population values.

$$\overline{X} \approx \bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{3.3}$$

$$s_x^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})}{n-1} \tag{3.4}$$

$$\sigma^2 \approx \frac{(N-1)s_x^2}{N} \tag{3.5}$$

Population parameter estimates depend entirely on which sample, out of all possible samples, was selected in the experiment. To address this, statistical procedures have been developed to judge the quality of an estimated parameter.

Suppose the mean for each possible sample of size n of our population (totaling $\frac{N!}{n!(N-n)!}$ possible samples) was calculated and plotted as a histogram (Figure 3.1). The distribution of these sample means (sampling distribution) has a variance $Var(\bar{x})$ (sampling variance) and a mean (sampling mean) that is equivalent[1] to $\overline{X}$.

Like σ^2, directly computing $Var(\bar{x})$ is too expensive but can be accurately estimated.[2]

[1]Assuming no measurement error, which is a valid assumption given our simulation technique.

[2]This estimation again assumes no measurement error, as well as a sample size greater than 30.

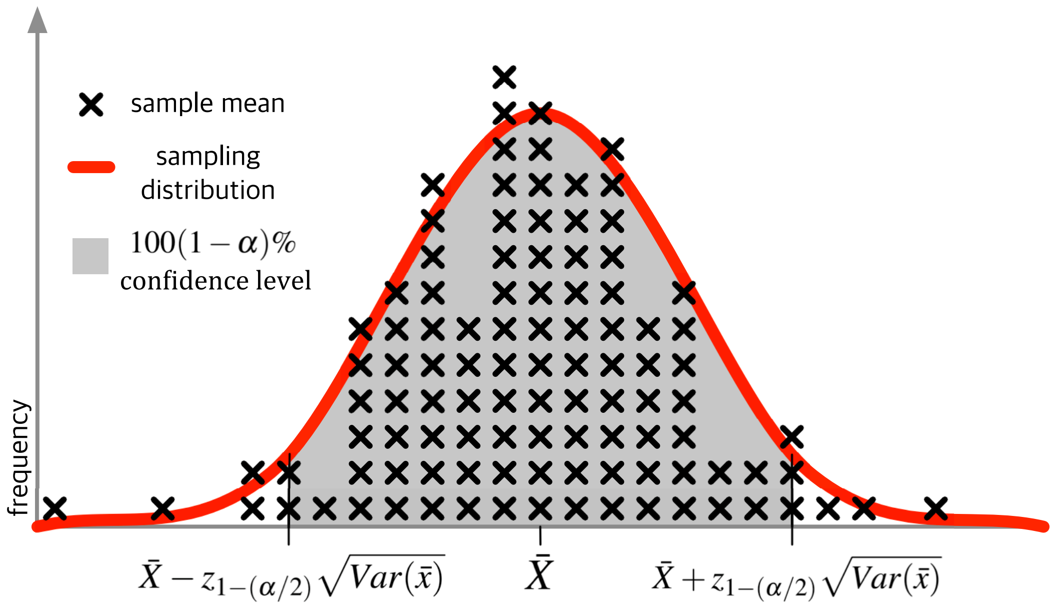


Figure 3.1: Theoretical Sampling Distribution

$$Var(\bar{x}) \approx \frac{s_x^2(N-n)}{Nn} \tag{3.6}$$

Once an estimator and its estimated accuracy have been computed, we can use normal theory to obtain approximate confidence intervals under a given confidence level $(1-\alpha)$ for the unknown parameter being estimated. The constant $z_{1-(\alpha/2)}$ is the $100[1-(\alpha/2)]$th percentile of the standard normal distribution.

$$\bar{x} \pm z_{1-(\alpha/2)}\sqrt{Var(\bar{x})} \tag{3.7}$$

A confidence interval interpretation is if n elements are sampled from a population repeatedly, with a given sampling strategy, $100[1-(\alpha/2)]\%$ of each sample's confidence interval would include the true (but unknown) population parameter.

A critical assumption of confidence intervals is of normality, or that the sampling distribution is Gaussian in shape. Fortunately, the central limit theorem of statistics guarantees that for large enough sample sizes $(n > 30)$, sampling distributions tend to be normal, regardless of the underlying distribution of the element characteristics in the sample.[3]

[3]This guarantee of normality is only for linear estimators (e.g. a mean estimator).

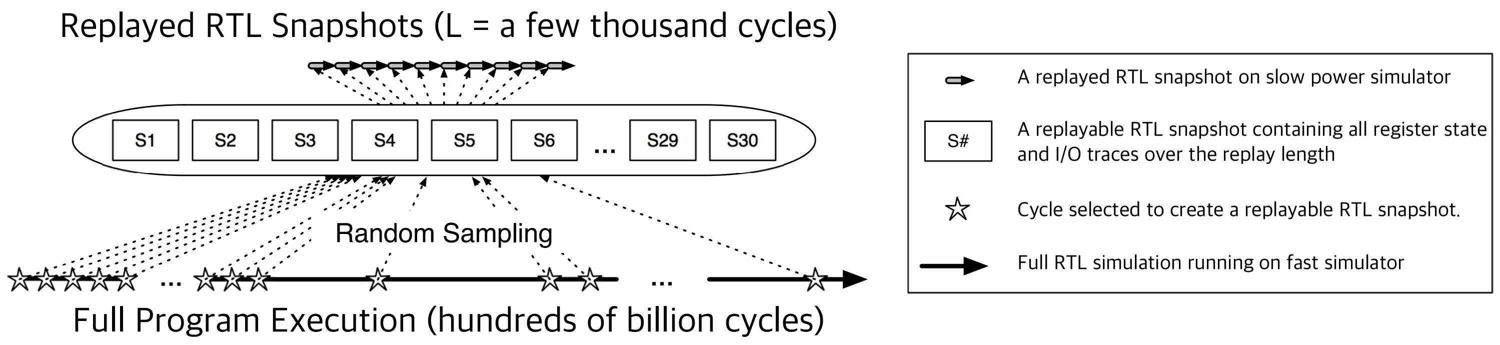

Replayed RTL Snapshots (L = a few thousand cycles)
S1
S2
S3
S4
S5
S6
...
S29
S30
Random Sampling
...
...
Full Program Execution (hundreds of billion cycles)
A replayed RTL snapshot on slow power simulator
S#
A replayable RTL snapshot containing all register state and I/O traces over the replay length
Cycle selected to create a replayable RTL snapshot.
Full RTL simulation running on fast simulator

Figure 3.2: Sample-Based Energy Simulation Methodology

In other words, given random sampling, enough samples, and no measurement error, calculated confidence intervals are *always* representative of the accuracy of an estimator.

To determine the minimum sample size, the previous equations can be analyzed to derive the following approximate relationship, where ϵ represents the maximum relative difference allowed between the estimated parameter and the unknown true population parameter.

$$n \geq \max\left\{\left(\frac{z_{1-(\alpha/2)}^2 s_x^2}{\epsilon^2 \bar{x}^2}\right), 30\right\} \tag{3.8}$$

By using this equation, we can validate whether our sample size was large enough to give adequate accuracy.

3.2 Methodology Overview

Our sample-based RTL energy simulation methodology quickly and accurately estimates both performance and power of long running applications on arbitrary hardware designs. This methodology obtains random sample points from a fast simulator and replays them on a slow but detailed simulator. Figure 3.2 shows the basic idea behind our methodology.

First, a design's performance is evaluated by an accelerated full-system RTL simulation, during which a set of replayable RTL snapshots is obtained. A replayable RTL snapshot, at cycle c, of a given replay length L, consists of all information necessary to replay from c to $c+L$ on a very slow but extremely detailed gate-level simulation. More specifically, a replayable RTL snapshot contains all RTL state at cycle c and a trace of all I/O signals of length L starting at cycle c. As an optimization, the I/O traces of a given replayable RTL snapshot are read out from the simulation only when the next replayable RTL snapshot is picked.

We can obtain the best statistical properties when the replayable RTL snapshots are randomly captured over the course of the program's execution (Section 3.1). Since knowing the length of a full program execution is impossible a priori, we employ reservoir sampling [45] to address this problem. With this algorithm and a desired sample size n, the first n replayable RTL snapshots are recorded with the sample size. The kth element where $k > n$ is recorded with the probability of n/k, and then randomly replacing one of the existing replayable RTL snapshots. Note that the probability of selection decreases with longer execution, thus diminishing the sampling overhead. At the end of the program execution, we have n replayable RTL snapshots that were selected at random, without replacement. As seen in 5.2, the simulation time of very long-running applications with sampling is very close to the simulation time without sampling.

In order to replay each replayable RTL snapshot, the RTL state is loaded into the detailed simulator. For each cycle in the replay, the inputs from the I/O trace are fed to the input of the target design, and outputs are verified against the output values of the design. Note that unlike the previous simulation sampling techniques [36, 13, 37], there is no state warming problem due to the exactness of the replayable RTL snapshot. In addition, all replayable RTL snapshots are independent, so we can parallelize their replays on multiple instances of the detailed simulator.

To estimate power, the detailed simulator is a gate-level simulation of the given RTL design. The simulation computes the signal activities of the gate-level design, accounting for detailed timing from floorplanning, placement and routing. An industrial power analysis tool computes the power of each replayable RTL snapshot from the detailed signal activities. By aggregating the power of all replayable RTL snapshots, we can predict the average power and corresponding confidence interval of a full execution of benchmarks. In general, the derived confidence intervals are very small with a small number of replayable RTL snapshots and 99.9% confidence, regardless of the length of simulation.

Chapter 4

The Strober Framework

In this chapter, we describe the Strober framework, our implementation of the sampling-based energy-modeling methodology for Chisel RTL designs. In Section 4.1, we briefly introduce Chisel. Next, in Section 4.2 we explain how any hardware design written in Chisel is automatically transformed into a FAME1 simulator with simulation snapshot capture capability. In Section 4.4, we explain how RTL snapshots are replayed on gate-level simulation using commercial CAD tools that are industry-standard and widely available to academics through academic licensing programs. We also explain how to estimate DRAM's power consumption using activity counters in Section 4.5. Lastly, a simple analytic performance model for the Strober framework is introduced in Section 4.6.

4.1 Chisel

Chisel [16] is a hardware construction language embedded in Scala [46] that helps hardware designers generate RTL with various parameters by providing access to advanced parameterization systems. Note that Chisel is not a high-level synthesis tool; like Perl or Python scripts that modify or generate Verilog [47], a designer uses Chisel's host language Scala to create and connect structural RTL components. Chisel can also generate fast C emulators and high-level simulation interfaces for a design.

Most importantly, Chisel's backend provides an intermediate representation that can be manipulated by custom transforms. Our toolchain includes custom transforms and platform-specific hardware generators that automatically convert Chisel designs to FPGA simulators, as explained in Section 4.2.

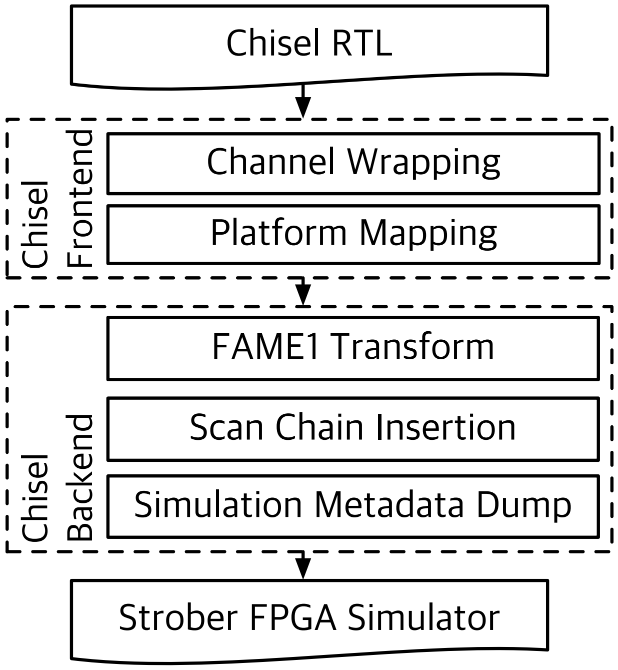


Figure 4.1: Flow to Generate FPGA Simulators

4.2 FPGA RTL Simulation

Manually writing accurate and fast FPGA simulators is very difficult and tedious. Instead, Strober automatically transforms the RTL design into a FAME1 simulator that can be mapped onto an FPGA. Figure 4.1 shows the tool flow to automatically generate FPGA RTL simulators. The following subsections describe the strategies and techniques required to enable this automatic transformation.

4.3 Simulation Mapping

Strober creates a FAME1 simulator [17] from a given hardware design, similar to token-based timing simulators manually implemented in previous work [48, 49, 50, 51, 52]. Note that these simulators are instances of synchronous dataflow [53].

Figure 4.2 depicts an automatic FAME1 transformation on an arbitrary RTL design to generate a token-based simulator of that design. Communication channels wrap a simulation module to buffer timing tokens from other

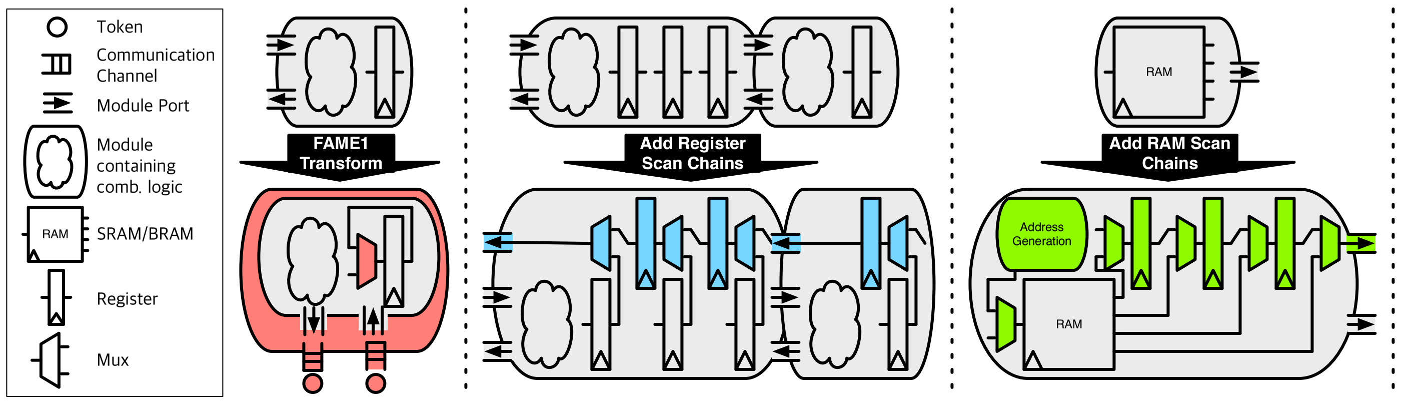


Figure 4.2: Transform from RTL Designs to FPGA RTL Simulators

simulation modules. A globally enabled mux is also added before each register allowing it to capture its own output, thus enabling the entire simulation module to stall when the global enable is not set. A running simulation module will stall if the input buffer is empty or the output buffer is full. When all input timing tokens are ready for a given cycle and output buffer space is available for all output timing tokens, a simulation module fires, consuming input tokens, simulating one cycle, and generating output tokens.

This FAME1 transformation allows simulation modules to run decoupled, which is an important optimization when not all components can be hosted on a single FPGA. In our case studies discussed in Section **??**, the main memory and I/O devices are mapped to the host platform memory and the software components respectively, while the RTL designs are mapped to the FPGA fabric.

The Strober framework flow, described in Figure 4.1, contains this FAME1 transformation implemented in compiler passes as well as libraries written in Chisel. Channel wrapping, implemented as a Chisel library, systematically generates communication channels for all I/O ports in a simulation module, connecting them properly. This process also adds I/O trace buffers for each channel for I/O recording, which is required for replayable RTL snapshots (Section 4.3.1). To complete FAME1 transforms, a global enable signal is connected by traversing all state elements in the Chisel backend.

4.3.1 State Snapshotting using Scan Chains

The sample-based energy simulation methodology described in Chapter 3 requires that replayable RTL snapshots are captured during the FPGA simulation. These replayable snapshots include all register and SRAM values, which can be a large amount of data in complex designs. This constraint requires 1) an efficient implementation to read a large amount of data from an FPGA, and 2) a systematic and automatic approach to transform an arbitrary design. Strober adds scan chains in the Chisel backend (Figure 4.1) to meet these requirements.

Figure 4.2 shows a basic scan chain to read the values from registers. Immediately after the simulation stalls to create the replayable RTL snapshot, the scan chain registers capture the RTL state in the scan chain. All register state can then be read via the scan chain.

Due to RAMs' large capacity, basic scan chains cannot be used. Moreover, we cannot change the number of RAMs' ports since they need to be mapped to Block RAMs on an FPGA. To address this issue, Strober adds special scan chains for RAMs (Figure 4.2). When the simulation stalls, the scan chain generates the address to be read, and copies the data from the read port. After all the data in the scan chain is read out, the scan chain generates the next address, and copies the data again. This process repeats until all necessary data is read out.

4.3.2 Platform Mapping

To enable a design to be mapped to many different FPGA products, the Strober framework has a platform-mapping transformation that automatically generates the correct interface for a specific FPGA platforms. This transformation (Figure 4.1) generates a wrapper to convert platform-specific data to simulation timing tokens, as well as assigns addresses for the communication channels and scan chain outputs. Simulation meta data for the simulation software driver is also dumped in the custom transformation. We currently support Xilinx Zynq boards for the host platform but plan to support more platforms in the future.

4.4 Replaying on Gate-level Simulation

The FPGA RTL simulators generated by the Strober framework provide cycle-exact performance estimates, but the replayable RTL snapshots must be simulated on a gate-level simulator to compute average power. Figure 4.3 shows the tool flow to replay samples on gate-level simulation. The Chisel Verilog backend generates Verilog RTL from Chisel RTL for the ASIC tool flow. Next, a synthesis tool[1] and place-and-route tool[2] produce the gate-level netlist and the post-layout design, respectively. Gate-level simulation [3], with very detailed timing, simulates the post-layout design to compute signal activities.

Replayable RTL snapshots are obtained from the Strober-generated FPGA RTL simulator, as explained in Section 4.2. The RTL state is loaded into the gate-level simulation, and the input traces are fed to the inputs of the design. Moreover, *the output values of the design are compared with the output traces, which ensures samples are replayed correctly.* Samples are independent of one another, so we can replay them on multiple instances of gate-level simulation in parallel.

The generated signal activities are consumed by the power analysis tool[4]

[1]For synthesis, we used Design Compiler ® J-2014.09-SP4.

[2]For place-and-route, we used IC Compiler ™ J-2014.09-SP4.

[3]For gate-level simulation, we used VCS ® H-2013.06.

[4]For power analysis, we used PrimeTime ® PX J-2014.12-SP2.

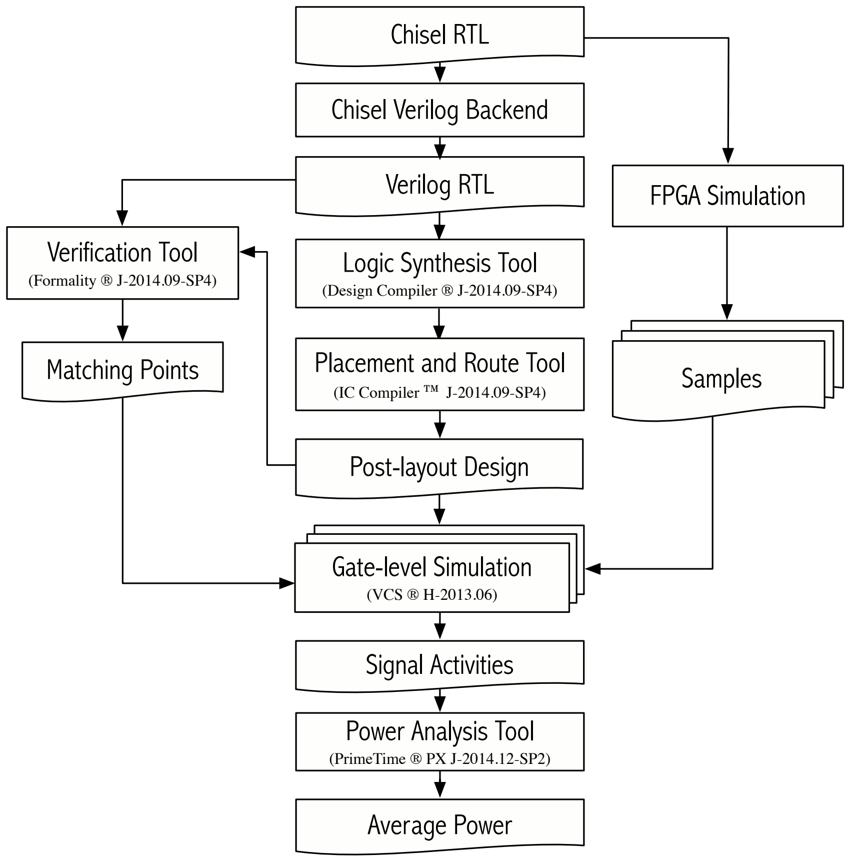


Figure 4.3: Sample Replay Flow

to estimate total power consumption for that replayable RTL snapshot. By calculating the mean of each power result, we can obtain the average power of all replayable RTL snapshots. As explained in Section 3.1, this average is an accurate estimation of the total application's power consumption on the given RTL design.

However, there are three key challenges to replay samples on gate-level simulation, addressed in the following subsections.

4.4.1 Signal Name Mangling in the Gate-level Netlist

One difficulty in initializing the RTL state is that register signal names are mangled by the optimizations performed by CAD tools. Because we cannot use the RTL signal names to load the state snapshots on gate-level simulation, we use a commercial formal verification tool[5] to match nodes between RTL designs and gate-level netlists (Figure 4.3).

The synthesis tool generates information about optimizations applied to a designs to help formal verification. By using this information, the formal verification tool first finds the matching points between RTL and the gate-level design (including registers) and then verifies the equality of the two designs. The matching results of this tool enable us to construct a name mapping table and translate FPGA RTL names into gate-level netlist names.

4.4.2 State Snapshot Loading on Gate-level Simulation

To load the register values into the gate-level simulation, we originally translated the values into scripts that were read by our commercial Verilog simulator. Unfortunately, this simulator could only execute 400 commands per second, which for a design of 35k flip-flops with 30 replayable RTL snapshots takes 40 minutes to load. While this is unacceptably slow for Strober's framework, writing a customized testbench for each design configuration is very cumbersome and error-prone.

We address this issue by writing a custom state snapshot loader that uses the Verilog Programming Language Interface [54]. The commercial Verilog simulators are compiled with this loader, which handles the snapshot loading commands efficiently. With this implementation, gate-level simulation can handle 20000 commands per second, reducing runtime to only 54 seconds for 30 samples with the example in-order processor.

4.4.3 Register Retiming

Another big challenge in loading state snapshots is handling register retiming. Register retiming is a technique to move datapath registers, reducing the critical path, area, or both [55]. For example, RTL designers often depend on this technique for writing floating-point units (FPUs), relying on CAD tools to automatically balance the stages in a datapath pipeline. Unfortunately, we cannot easily reconstruct the values of retimed registers from the RTL state snapshot.

Instead, we can capture the I/O values of the retimed datapath. First, note the retimed datapaths are annotated by the designers with the desired

[5]For formal verification, we used Formality ® J-2014.09-SP4. Again, all commercial CAD tools used are industry-standard and widely available to academics through academic licensing programs.

latency. For the n-cycle-latency datapath, a custom transform adds shift registers which capture the I/O values for the last n cycles (and the corresponding scan chains). The I/O signals of the retimed datapaths are forced externally in the simulation for n cycles before loading the snapshots to recover their internal state. By starting replays at this point, we can simulate each sample snapshot with fully-recovered state.

4.5 DRAM Power Estimation

DRAM power consumption is affected by the DRAM's internal operations (which can be triggered by memory access requests) and its internal state. For example, DRAM's internal read and write operations trigger data transfer through DRAM's I/O bus, causing dynamic power consumption. However, knowing the physical address mapping, the DRAM controller's policies, and all memory access requests is enough to predict any given DRAM's internal operations, and thus predict its power consumption. As in the experimental settings specified in Kim et al. [56], we use Micron's LPDDR2 SDRAM S4 [57] with eight banks, and 16K (16×1024) rows for each bank. We assume a bank-interleaved memory mapping where adjacent memory addresses are distributed across different banks. Finally, we assume an open-page policy, where DRAM banks are kept active after a row access.

To capture the DRAM memory requests, we attach counters to the memory request output ports. Using the known memory mapping, the physical address of each memory request is translated into the bank number and the row number. The previously accessed row and bank numbers are stored with the counter data to enable determining whether the row activation operation will occur. From the counter values, we know the number of read/write operations and the number of row activation operations. With this information and DRAM configurations, the DRAM power can be calculated using a spreadsheet power calculator provided by Micron [58].

4.6 Simulation Performance Model

To demonstrate the opportunity for significant speedup over the existing CAD tools, we present a simple analytic performance model of the Strober framework in this section. To estimate the overall time, we should consider (1) the synthesis time for the FPGA simulator, (2) the FPGA simulation time, (3) the ASIC tool chain time (logic synthesis, placement, routing, and formal verification), and (4) the replay time for sample snapshots. Note that (3) is independent from (1) and (2), so the overall time is expressed as follows:

$$T_{overall} = max(T_{FPGAsyn} + T_{FPGAsim}, T_{ASIC}) + T_{replay}$$

The ASIC tool chain time, T_{ASIC}, tends to be long for complex designs. However, we run very long-running application on the FPGA simulator, thus resulting in $T_{ASIC} < T_{FPGAsyn} + T_{FPGAsim}$. In this paper, the synthesis time for the FPGA simulator, T_{FPGA_syn}, can be up to one hour with a two-way out-of-order processor while T_{ASIC} is around three or four hours. Also note that $T_{FPGAsyn} \ll T_{FPGAsim}$ for real-world long-running applications.

To estimate $T_{FPGAsim}$, assume the FPGA simulation runs at K_f Hz. Let N and L be the total simulation cycles and the replay length respectively. Reservoir sampling [45] ensures that the number of elements recorded during the simulation is roughly $2nln((N/L)/n))$ with the sample size n. The FPGA simulation time, $T_{FPGAsim}$, is therefore:

$$T_{FPGAsim} = T_{run} + T_{sample} \approx N/K_f + T_{rec} \times 2nln(N/nL)$$

where T_{run}, T_{sample}, T_{rec} are the simulation running time, the total sampling time, and the time to read out a single replayable RTL snapshot, respectively.

T_{replay} is decomposed into (1) the snapshot loading time, (2) the snapshot replay time, and (3) the power analysis tool time. The snapshot loading time is considered because it can be very slow without a proper implementation (Section 4.4.2). For the snapshot replay time, suppose the gate-level simulation runs at K_g Hz. In addition, only L cycles are replayed for each sample snapshot. We provide the switching activity interface format (SAIF) files to the power analysis tool for the average power of each sample snapshot, and thus, the power analysis time is independent of the length of each sample snapshot. Lastly, as explained in Chapter 3, each replays in the sample are independent and can be parallelized. Therefore, assuming P instances of gate-level simulation, the total replay time is:

$$T_{replay} = \frac{n \times (T_{load} + (L/K_g) + T_{power})}{P}$$

where T_{load} is the time to load each RTL state into the gate-level simulation, and T_{power} is the time to run the power analysis tool for a single sample snapshot.

For the example two-way out-of-order processor used in this paper, the FPGA synthesis time with Strober was around one hour[6], the FPGA simulation runs at 3.6 MHz, and the gate-level simulation runs at 12 Hz. In

[6]For FPGA synthesis, we used Vivado ® 2014.4.

addition, the recording time per replayable RTL snapshot is 1.3 seconds, the sample loading time on gate-level simulation is 3 seconds, and the time for power analysis[7] is around two and a half minutes. Suppose we simulate a benchmark whose execution length is 100 billion cycles on the two-way out-of-order, has a sample of 100 replayable RTL snapshots (with replay length of 1000 cycles), on 10 instances of gate-level simulation. Plugging these numbers to the equations, we can calculate the overall simulation time:

$$T_{FPGAsyn} = 3600\ s$$

$$T_{run} = \frac{10^{11} cycles}{3.6 \times 10^6 Hz} = 27778\ s$$

$$T_{sample} = 1.3 \times 100 \times 2 \times ln(\frac{10^{11}}{100 \times 10^3})) = 3592\ s$$

$$T_{replay} = \frac{100 \times (10^3 cycles/12Hz + 150)}{10} = 2333\ s$$

Thus, $T_{overall} = T_{run} + T_{sample} + T_{replay} = 33703$ seconds or 9.4 hours. Note that it will take $10^{11} cycles/300KHz = 3.86$ days even on fast microarchitectural software simulators and
$10^{11} cycles/12Hz = 264$ years on gate-level simulation!

[7]For power analysis, we again used PrimeTime ® PX J-2014.12-SP2.

Chapter 5

Evaluation

In this chapter, the Strober framework is evaluated and validated. The target hardware designs, Rocket-chip[18] and BOOM[19], used in the evaluation are introduced in Section 5.1. Although traditional in-order and out-of-order processors are used in the evaluation, note that the Strober framework is not limited to these designs and thus can be used for any novel hardware designs. Next, the simulation performance of the Strober framework is measured and how we can further improve its performance is explained in Section 5.2. In Section 5.3, the DRAM timing model in the Strober framework is validated. Finally, the power estimation of the Strober framework is validated with the RISC-V benchmarks in Section 5.4.

5.1 Target Designs

To demonstrate Strober's ability to augment arbitrary Chisel RTL, we evaluated two different synthesizable open-source cores, both which leverage the open-source *Rocket-Chip* SoC generator [18]. The first core is *Rocket*, a 5-stage single-issue in-order core. The second core is *BOOM*, a parameterized, superscalar out-of-order core [19]. Both cores implement the full 64-bit scalar RISC-V ISA, which includes support for atomics, IEEE 754-2008 floating-point, and page-based virtual memory. Note that the Strober framework is built upon commercial CAD tools, which report accurate timing and area for RTL designs. Figure 5.1 shows a sample floorplan of the two-way superscalar out-of-order processor. For this evaluation, both cores were simulated at 1 GHz frequency, however silicon implementations of Rocket have been demonstrated to reach 1.3 GHz [59] and 1.65 GHz [60] in an IBM 45nm SOI technology.

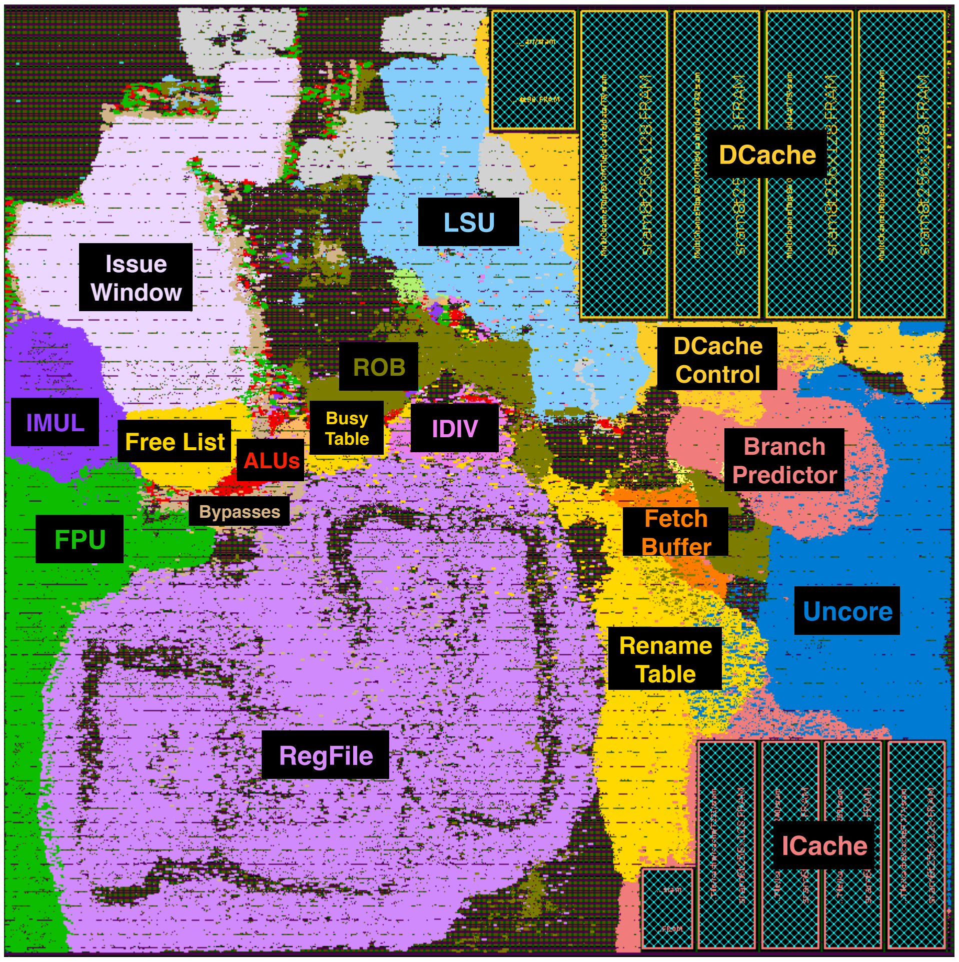


Figure 5.1: Floorplan of BOOM-2w

	Rocket	**BOOM-1w**	**BOOM-2w**
Fetch-width	1	1	2
Issue-width	1	1	2
Issue slots	-	12	16
ROB size	-	24	32
Ld/St entries	-	8/8	8/8
Physical registers	32(int)/32(fp)	100	110
L1 I$ and D$	16KiB/16KiB		
DRAM latency	100 cycles		
Technology	TSMC 45nm		

Table 5.1: Processor Parameters

5.2 Simulation Performance

	LinuxBoot	**Coremark**	**gcc**
Simulation Cycles (10^9)	0.5	3.92	73.39
Record Counts	980	1116	1497
Simulation Time with Sampling (min)	12.88	32.80	344.00
Simulation Time without Sampling (min)	3.68	11.00	312.25

Table 5.2: Simulation Performance Evaluation for Each Benchmark on the Two-way BOOM Processor

For Rocket Chip target systems running under Strober, target I/O devices are mapped to software on the host CPU, not the FPGA, causing a communication overhead that stalls the simulator every 256 cycles. The target simulator is also stalled while capturing a replayable RTL snapshot.

Table 5.2 shows the performance evaluation of Strober with the two-way BOOM processor running long benchmarks showed in 6.1. The record counts, the number of sample recording during each simulation run, only moderately increases as explained by reservoir sampling. Therefore, the sampling overhead is very small for long-running simulations.

For the `gcc` runs of 70 billion cycles, Strober achieved a simulation speed of around 3.56 MHz. For comparison, the unmodified Rocket and BOOM cores

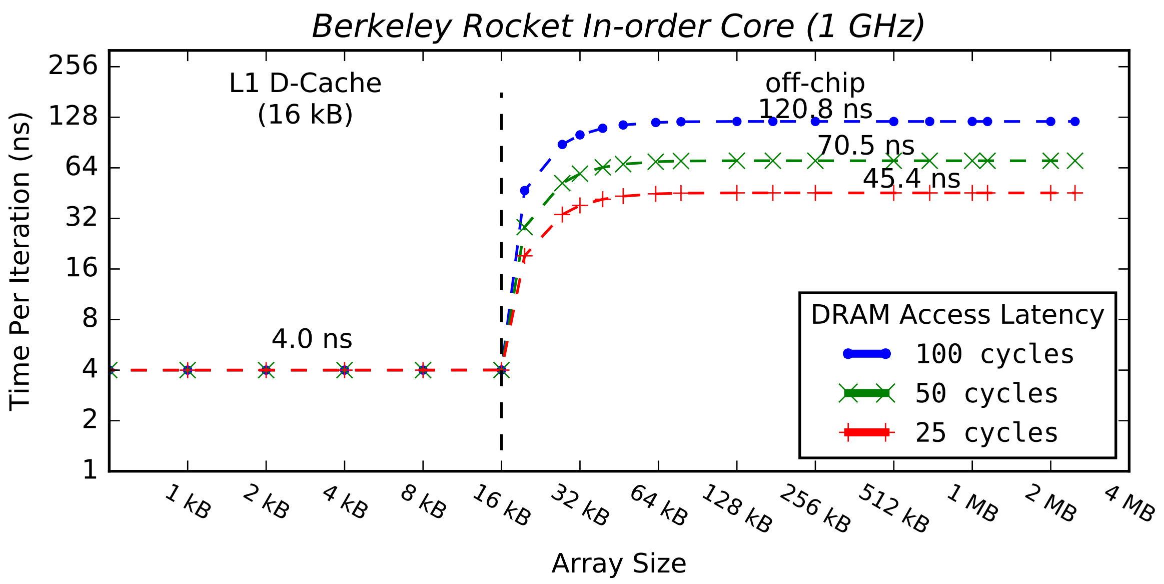


Figure 5.2: DRAM Timing Model Validation. A pointer-chase through increasing sizes of arrays demonstrates the load-to-load latency of different levels of the memory hierarchy. By varying the simulated DRAM latencies for the Rocket-chip processor, a change in the off-chip latency can be observed.

both can be synthesized at 50 MHz on the same zc706 FPGA.

Note that the simulation speed in this evaluation is not the fundamental limitation of the Strober framework. The reason Strober is much slower than the FPGA emulation is the tethered systems are simulated. In the tethered systems, the target hardware designs are connected to the frontend server in software over decoupled I/O so that the frontend server can handle system calls and check the end of the program execution. Thus, the frontend sever initiates the program execution by loading executable binaries, and is polling during the execution even though there are no special activities in the target design, thus slowing down the simulation significantly. We can resolve this issue by reducing the frequency of the polling, and eventually, simulating stand-alone target systems.

5.3 DRAM Timing Model Validation

One of the significant challenges of simulating RTL designs on FPGAs is properly modeling the interfaces to the outside environment. In simulating the processor designs discussed in Section 5.1, the DRAM behavior must be properly

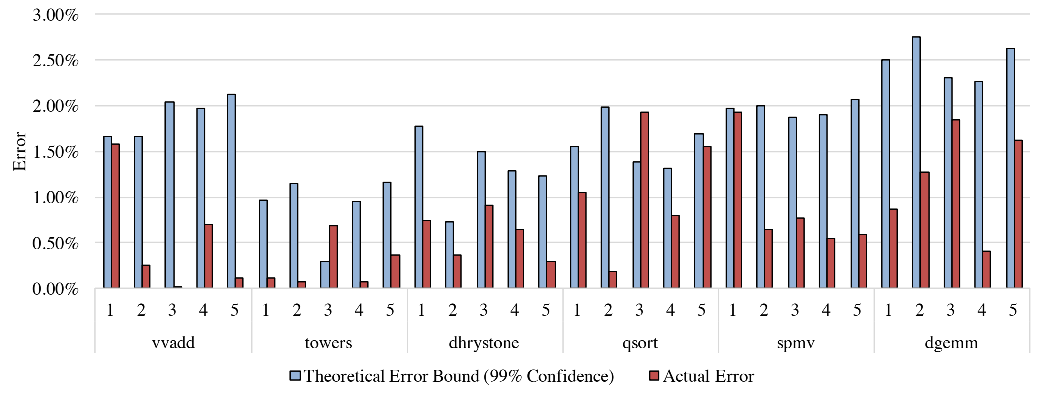


Figure 5.3: Comparison for the Theoretical Error Bounds with the Actual Errors

modeled. Figure 5.2 demonstrates Strober's ability to modify the simulated DRAM latency. Repeated measurements of a pointer-chase benchmark [61] is used to measure the L1-cache size and memory access latencies to the L1 cache and off-chip DRAM.

5.4 Power Validation

Benchmark	**Simulated Cycles**	**Replayed Cycles**	**Coverage**
vvadd	200521	30×128	1.92%
towers	410752	30×128	0.93%
dhrystone	396790	30×128	0.97%
qsort	187160	30×128	2.05%
spmv	927144	30×128	0.41%
dgemm	1833075	30×128	0.21%

Table 5.3: Simulated and Replayed Cycles for Each Benchmark on the Rocket Processor

To validate our Strober framework and the sample-based RTL energy modeling methodology, we run the microbenchmarks included in the Rocket-Chip framework to completion on a gate-level-simulation of Rocket. The switching activity for the entire benchmark is used to calculate the actual average power. Also, we obtain 30 random sample snapshots of 128 cycles from the FPGA simulation, and by replaying these, we calculate the average power as well as their error bounds with 99% confidence. Then, we compare those error bounds over the actual errors as in Figure 5.3. We repeated this process five times for each benchmark.

Note that even though the samples cover only less than 2.1% of the cycles as shown in Table 5.3, the errors tend to be very small. Moreover, in most cases, the actual errors are within the error bounds computed from the samples. This also shows that *the errors are independent of the length of execution.* While the third sampling of *towers*, and the third of *qsort* are slightly outside their error bounds, this result is somewhat expected due to the probabilistic nature of statistical sampling. Nevertheless, their actual errors are still very small, less than 2%.

Chapter 6

Case Study

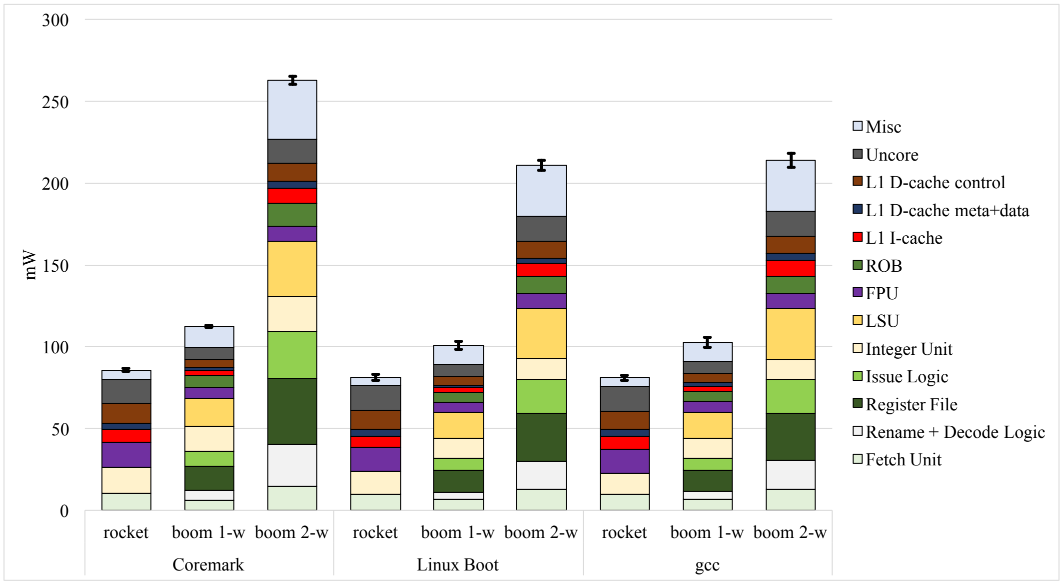
In this chapter, we demonstrate the capability of the Strober framework with real-world hardware designs and software benchmarks. The benchmarks used in this case study are introduced in Section 6.1. Next, the performance, power, and energy estimates of these benchmarks with Rocket and BOOM are presented in Section 6.2.

6.1 Benchmarks

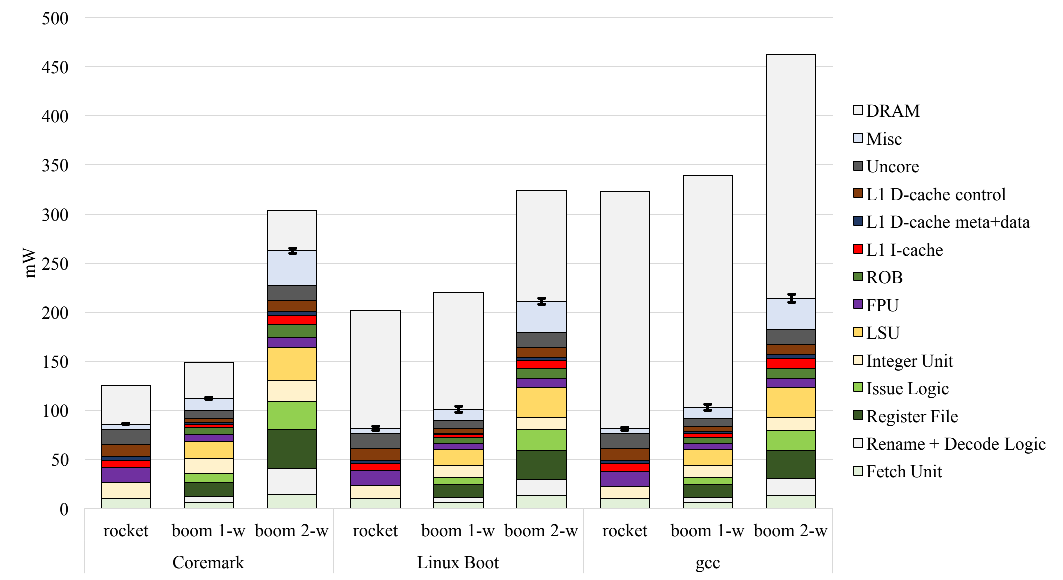
We chose three disparate workloads to demonstrate Strober's ability to measure target design performance, power, and energy usage. The first is CoreMark, a benchmark designed to stress processor pipelines [62]. The second workload boots the RISC-V port of Linux on a small BusyBox disk image, executes the `uname` and `ls` commands, and then powers down. The third workload executes the SPECint benchmark `403.gcc`[63] on Linux. For `gcc`, we execute the first 20B instructions (or 20%) of the SPECint reference input workload "gcc 166.in".

6.2 Performance, Power, and Energy Analysis

Figure 6.1 compares the energy breakdown of the Rocket, BOOM-1w, and BOOM-2w cores using 30 random sample snapshots for each benchmark. The performance differences between the cores is easiest to see when running CoreMark, a small benchmark designed to fit in L1 caches and stress processor's integer pipelines. BOOM-1w is 9.8% faster than Rocket, and BOOM-2w is 58% faster. However, BOOM-2w uses 3× the power, while Rocket is the most energy-efficient (Figure 6.2).



(a) Power Breakdown without DRAM Power



(b) Power Breakdown with DRAM power

Figure 6.1: Power Breakdown with Error Bounds using 30 Random Samples for Each Benchmark

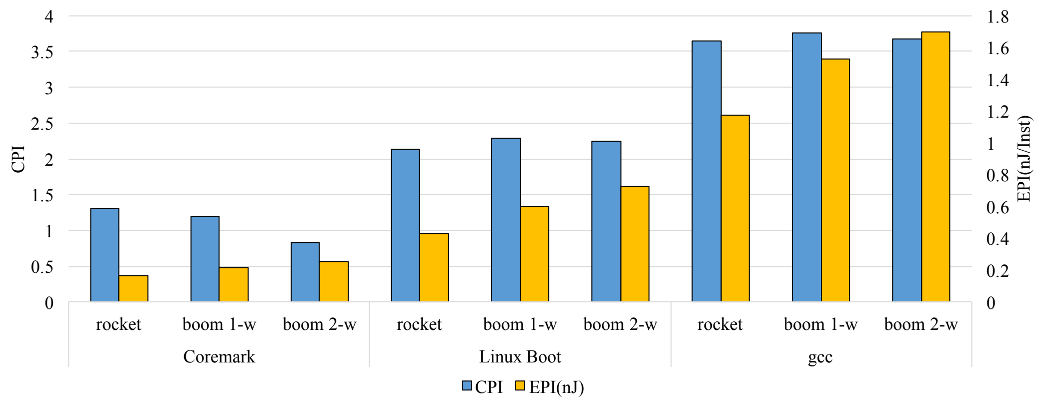


Figure 6.2: Performance and Energy Efficiency of the Rocket and BOOM Processors

The other benchmarks use a much larger memory footprint than CoreMark, as seen in the increased DRAM power usage. On Linux-boot, clock for clock, Rocket's shorter branch resolution latency allows it to outperform BOOM, which has only a simple branch predictor in the version used in this case study.

Details aside, this case study shows the validity of using Strober as a basis for design-space exploration in architecture research. With Strober, researchers now have the ability to run real programs on RTL with a full evaluation of energy, area, and performance. In addition, each sample snapshot contains a timestamp, so by using performance counters we can correlate performance and power at a specific point as shown in Figure 6.3. Using this case study as an example, the turn-around time for evaluating 70 billion cycles on BOOM-2w is approximately 7 hours for a complete evaluation. We believe this is fast enough to enable realtime feedback in the RTL design loop.

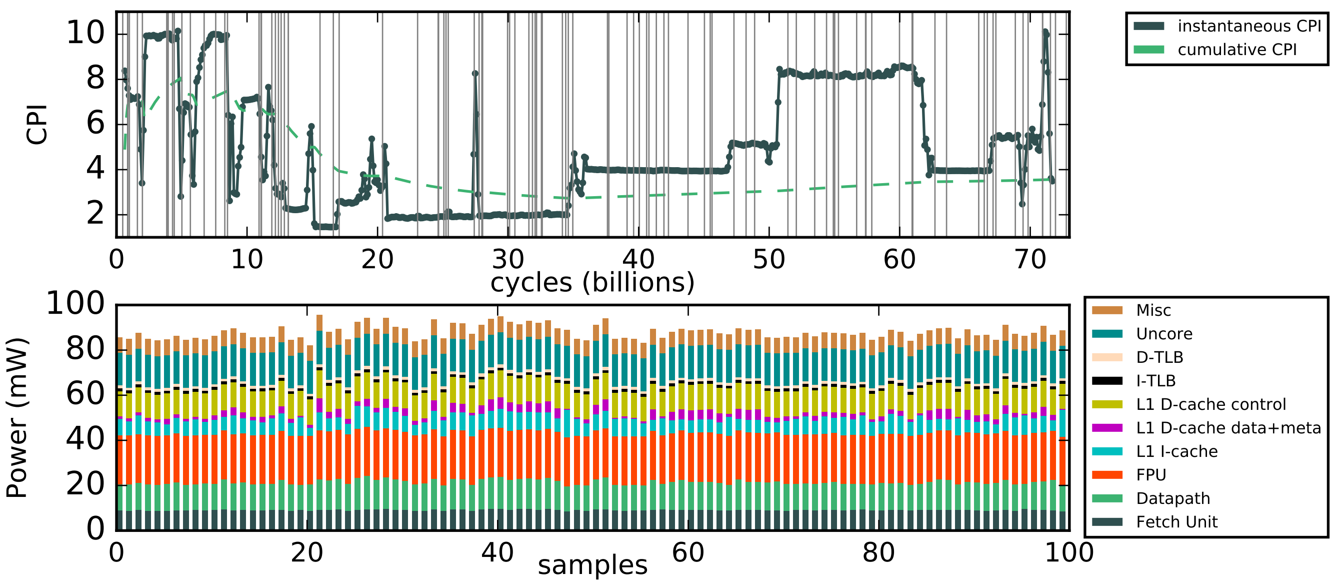


Figure 6.3: The CPI of the first 20B instructions (or 20%) of `403.gcc` as executed on Rocket. The CPI is sampled every 100M cycles by a separate user program running on Rocket. Grey vertical lines denote when a Strober snapshot was taken.

Chapter 7

Conclusion and Future Work

In this paper we presented a sample-based RTL energy modeling methodology that captures replayable RTL snapshots from a fast performance simulation and replays them on a detailed power simulation. We showed the statistical robustness of this methodology, including the ability to generate confidence intervals for any power prediction.

Next, we introduced Strober, a framework for taking existing RTL designs written in the Chisel hardware construction language, and generating a cycle-accurate, decoupled simulator that can be executed on an FPGA. The instrumented simulator can be used to not only measure the cycle-accurate performance of the RTL design, but to generate random RTL snapshots that can be replayed (in parallel) in a detailed gate-level simulator. We also demonstrated significant theoretical speedups using an analytical model for simulation performance. While this work demonstrates using Chisel RTL, the presented methodology is amenable to any hardware construction language that provides a facility for developing new hardware transformation passes.

We then validated our methodology and framework for simulation performance, and power accuracy. Finally, we demonstrated our framework by running three complex RTL designs through our toolchain to obtain timing, area, performance, and average power for a variety of benchmarks. These case studies serve as an example of how Strober can not only provide ground truth for building faster and more flexible abstract power models, but can in and of itself be a tool for design-space exploration at the RTL level.

To sum up, Strober is much faster than exiting micro-architectural software simulators and commercial verilog simulators. Also, its accuracy is very close to that of commercial CAD tools. Finally, it is an easy-to-use tool as FPGA simulators are automatically generated from the Chisel designs and the sample replay flow is easily pluggable to the commercial CAD tool flow. We also demonstrated the Strober's capability for real-world hardware designs and software applications.

Strober is an on-going project. This framework was originally developed based on Chisel 2. However, a main problem of Chisel 2 is its frontend APIs and backend passes are tightly coupled, thus making it difficult to implement and maintain custom transforms. Now, Chisel 2 evolves into Chisel 3 and FIRRTL [64]. The idea is FIRRTL provides the intermediate representation (IR) for hardware circuits so that hardware designers can easily write their own custom transforms, and Chisel 3 only emits the IR manipulated by the FIRRTL compiler. Strober is being reimplemented using Chisel 3 and FIRRTL, which makes the Strober framework more maintainable.

As porting Strober to Chisel 3 and FIRRTL, the ideas of FAME simulators presented in this thesis is generalized as MIDAS, a general FPGA simulation and debugging framework, along with a more realistic DRAM timing model. Indeed, Strober will be released as a use case of MIDAS for sample-based energy simulation [65]. Moreover, Strober will support assertions and printfs in FPGA simulation along with its snapshotting capability to ease debugging efforts with long-running applications. Last but not least, MIDAS will be the base for FireSim, the simulators for the Warehouse-Scale Computers (WSCs).

Bibliography

[1] B. Zimmer, Y. Lee, A. Puggelli, J. Kwak, R. Jevtic, B. Keller, S. Bailey, M. Blagojevic, P.-F. Chiu, H.-P. Le, P.-H. Chen, N. Sutardja, R. Avizienis, A. Waterman, B. Richards, P. Flatresse, E. Alon, K. Asanović, and B. Nikolic, "A RISC-V vector processor with tightly-integrated switched-capacitor DC-DC converters in 28nm FDSOI," in *VLSI*, 2015.

[2] E. Rotenberg, B. H. Dwiel, E. Forbes, Z. Zhang, R. Widialaksono, R. Basu Roy Chowdhury, N. Tshibangu, S. Lipa, W. R. Davis, and P. D. Franzon, "Rationale for a 3D heterogeneous multi-core processor," in *ICCD*, 2013.

[3] P. Pannuto, Y. Lee, Y.-s. Kuo, Z. Foo, B. Kempke, G. Kim, R. G. Dreslinski, D. Blaauw, and P. Dutta, "MBus : An Ultra-Low Power Interconnect Bus for Next Generation Nanopower Systems," in *ISCA*, 2015.

[4] X. Zhang, M. Lok, T. Tong, S. Chaput, S. K. Lee, B. Reagen, H. Lee, D. Brooks, and G.-y. Wei, "A Multi-Chip System Optimized for Insect-Scale Flapping-Wing Robots," in *VLSI*, 2015.

[5] J. Balkind, M. Mckeown, Y. Fu, T. Nguyen, S. Payne, X. Liang, M. Matl, and D. Wentzlaff, "OpenPiton : An Open Source Manycore Research Framework," in *ASPLOS*, 2016.

[6] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," in *ISCA*, 2000.

[7] N. Vijaykrishnan, M. Kandemir, M. J. Irwin, H. S. Kim, and W. Ye, "Energy-driven integrated hardware-software optimizations using SimplePower," in *ISCA*, 2000.

[8] S. Li, J. H. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *MICRO*, 2009.

[9] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "GPUWattch: enabling energy optimizations in GPGPUs," in *ISCA*, 2013.

[10] 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 *ISCA*, 2014.

[11] N. Binkert, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, D. A. Wood, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, and T. Krishna, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, Aug 2011.

[12] A. Patel, F. Afram, and S. Chen, "MARSSx86: A full system simulator for x86 CPUs," in *DAC*, 2011.

[13] T. F. T. Wenisch, R. R. E. R. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. J. Hoe, "SimFlex: Statistical Sampling of Computer System Simulation," *IEEE Micro*, vol. 26, pp. 18–31, Jul 2006.

[14] N. K. Choudhary, S. V. Wadhavkar, T. A. Shah, H. Mayukh, J. Gandhi, B. H. Dwiel, S. Navada, H. H. Najaf-abadi, and E. Rotenberg, "FabScalar: composing synthesizable RTL designs of arbitrary cores within a canonical superscalar template," in *ISCA*, 2011.

[15] D. Voitsechov and Y. Etsion, "Single-Graph Multiple Flows : Energy Efficient Design Alternative for GPGPUs," in *ISCA*, 2014.

[16] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: constructing hardware in a scala embedded language," in *DAC*, 2012.

[17] Z. Tan, A. Waterman, H. Cook, S. Bird, K. Asanović, D. Patterson, and D. P. Zhangxi Tan, Andrew Waterman, Henry Cook, Sarah Bird, Krste Asanović, "A Case for FAME: FPGA Architecture Model Execution," in *ISCA*, 2010.

[18] K. Asanović, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, S. Karandikar, B. Keller, D. Kim, J. Koenig, Y. Lee, E. Love, M. Maas, A. Magyar, H. Mao, M. Moreto, A. Ou, D. A. Patterson, B. Richards, C. Schmidt, S. Twigg, H. Vo, and A. Waterman, "The Rocket Chip Generator," Tech. Rep.

UCB/EECS-2016-17, EECS Department, University of California, Berkeley, Apr 2016.

[19] C. Celio, D. A. Patterson, and K. Asanović, “The Berkeley Out-of-Order Machine (BOOM): An Industry-Competitive, Synthesizable, Parameterized RISC-V Processor,” Tech. Rep. UCB/EECS-2015-167, EECS Department, University of California, Berkeley, Jun 2015.

[20] D. Kim, A. Izraelevitz, C. Celio, H. Kim, B. Zimmer, Y. Lee, J. Bachrach, and K. Asanović, “Strober : Fast and Accurate Sample-Based Energy Simulation for Arbitrary RTL,” in *ISCA*, 2016.

[21] F. Bellosa, “The benefits of event: driven energy accounting in power-sensitive systems,” in *The 9th ACM SIGOPS European workshop*, 2000.

[22] T. Li and L. K. John, “Run-time modeling and estimation of operating system power consumption,” in *SIGMETRICS*, 2003.

[23] C. Isci and M. Martonosi, “Runtime power monitoring in high-end processors: Methodology and empirical data,” in *MICRO*, 2003.

[24] W. Bircher, M. Valluri, J. Law, and L. John, “Runtime identification of microprocessor energy saving opportunities,” in *ISLPED*, 2005.

[25] W. L. Bircher and L. K. John, “Complete System Power Estimation: A Trickle-Down Approach Based on Performance Events,” in *ISPASS*, 2007.

[26] R. Bertran, M. Gonzelez, X. Martorell, N. Navarro, and E. Ayguade, “A Systematic Methodology to Generate Decomposable and Responsive Power Models for CMPs,” *IEEE Transactions on Computers*, vol. 62, pp. 1289–1302, Jul 2013.

[27] H. Shafi, P. J. Bohrer, J. Phelan, C. A. Rusu, and J. L. Peterson, “Design and validation of a performance and power simulator for PowerPC systems,” *IBM Journal of Research and Development*, vol. 47, no. 5.6, pp. 641–651, 2003.

[28] F. J. Mesa-Martinez, J. Nayfach-Battilana, and J. Renau, “Power model validation through thermal measurements,” 2007.

[29] S. L. Xi, H. Jacobson, P. Bose, G.-Y. Wei, and D. Brooks, “Quantifying sources of error in McPAT and potential impacts on architectural studies,” in *HPCA*, 2015.

[30] H. Jacobson, A. Buyuktosunoglu, P. Bose, E. Acar, and R. Eickemeyer, “Abstraction and microarchitecture scaling in early-stage power modeling,” in *HPCA*, 2011.

[31] D. Sunwoo, G. Y. Wu, N. a. Patil, and D. Chiou, “PrEsto: An FPGA-accelerated Power Estimation Methodology for Complex Systems,” in *FPGA*, 2010.

[32] A. Bhattacharjee, G. Contreras, and M. Martonosi, “Full-system chip multiprocessor power evaluations using FPGA-based emulation,” in *ISLPED*, 2008.

[33] J. Coburn, S. Ravi, and A. Raghunathan, “Power emulation: A new paradigm for power estimation,” in *DAC*, 2005.

[34] M. A. M. Ghodrat, K. Lahiri, and A. Raghunathan, “Accelerating system-on-chip power analysis using hybrid power estimation,” in *DAC*, 2007.

[35] D. Atienza, P. G. Del Valle, G. Paci, F. Poletti, L. Benini, G. De Micheli, and J. M. Mendias, “A fast HW/SW FPGA-based thermal emulation framework for multi-processor system-on-chip,” in *DAC*, 2006.

[36] R. Wunderlich, T. Wenisch, B. Falsafi, and J. Hoe, “SMARTS: accelerating microarchitecture simulation via rigorous statistical sampling,” *ISCA*, 2003.

[37] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe, “Statistical sampling of microarchitecture simulation,” *ACM Transactions on Modeling and Computer Simulation*, vol. 16, pp. 197–224, Jul 2006.

[38] E. K. Ardestani and J. Renau, “ESESC: A Fast Multicore Simulator Using Time-Based Sampling,” in *HPCA*, 2013.

[39] S. Hassani, G. Southern, and J. Renau, “LiveSim : Going Live with Microarchitecture Simulation,” in *HPCA*, 2016.

[40] E. S. Chung, M. K. Papamichael, E. Nurvitadhi, J. C. Hoe, K. Mai, and B. Falsafi, “PROTOFLEX: Towards Scalable, Full-System Multiprocessor Simulations Using FPGAs,” *ACM Transactions on Reconfigurable Technology and Systems*, vol. 2, pp. 1–32, Jun 2009.

[41] D. Chiou, D. Sunwoo, J. Kim, N. A. Patil, W. Reinhart, D. E. Johnson, J. Keefe, and H. Angepat, “FPGA-Accelerated Simulation Technologies (FAST): Fast, Full-System, Cycle-Accurate Simulators,” in *MICRO*, 2007.

[42] Z. Tan, A. Waterman, R. Avizienis, Y. Lee, H. Cook, D. Patterson, and K. Asanović, “RAMP Gold: An FPGA-based Architecture Simulator for Multiprocessors,” in *DAC*, 2010.

[43] M. Pellauer, M. Adler, M. Kinsy, A. Parashar, and J. Emer, “HAsim: FPGA-based high-detail multicore simulation using time-division multiplexing,” in *HPCA*, 2011.

[44] Z. Tan, Z. Qian, X. Chen, K. Asanović, and D. Patterson, “DIABLO: A Warehouse-Scale Computer Network Simulator using FPGAs,” in *ASPLOS*, 2015.

[45] J. S. Vitter, “Random sampling with a reservoir,” *ACM Transactions on Mathematical Software*, vol. 11, pp. 37–57, Mar 1985.

[46] M. Odersky, S. Micheloud, N. Mihaylov, M. Schinz, E. Stenman, M. Zenger, and et al., “An overview of the Scala programming language,” tech. rep., 2004.

[47] O. Shacham, M. Wachs, A. Danowitz, S. Galal, J. Brunhaver, W. Qadeer, S. Sankaranarayanan, A. Vassiliev, S. Richardson, and M. Horowitz, “Avoiding game over: Bringing Design to the Next Level,” DAC ’12, 2012.

[48] J. Emer, P. Ahuja, E. Borch, A. Klauser, S. Manne, S. Mukherjee, H. Patil, S. Wallace, N. Binkert, R. Espasa, and T. Juan, “Asim: A Performance Model Framework,” *Computer*, no. 2, pp. 68–76, 2002.

[49] G. Gibeling, A. Schultz, and K. Asanović, “The RAMP Architecture & Description Language,” in *The 2nd Workshop on Architecture Research using FPGA Platforms*, 2006.

[50] D. Chiou, W. H. Reinhart, and D. Eric Johnson, “The FAST methodology for high-speed SoC/computer simulation,” in *ICCAD*, 2007.

[51] M. Pellauer, M. Vijayaraghavan, M. Adler, Arvind, and J. Emer, “A-Port Networks: Preserving the Timed Behavior of Synchronous Systems for Modeling on FPGAs,” *ACM Transactions on Reconfigurable Technology and Systems*, vol. 2, pp. 1–26, Sep 2009.

[52] A. Khan, M. Vijayaraghavan, S. Boyd-Wickizer, and Arvind, “Fast and cycle-accurate modeling of a multicore processor,” in *ISPASS*, 2012.

[53] E. Lee and D. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.

[54] S. Sutherland, *The Verilog PLI Handbook*. The International Series in Engineering and Computer Science, Kluwer Academic Publishers, 2nd ed., 2002.

[55] C. E. Leiserson and J. B. Saxe, "Optimizing synchronous systems," in *22nd Annual Symposium on Foundations of Computer Science*, 1981.

[56] H. Kim, D. Broman, E. A. Lee, M. Zimmer, A. Shrivastava, and J. Oh, "A predictable and command-level priority-based DRAM controller for mixed-criticality systems," in *RTAS*, 2015.

[57] Micron Technology, "Micron mobile LPDDR2 SDRAM s4," datasheet, Micron Technology, Mar. 2012.

[58] Micron Technology, "Mobile LPDDR2 system-power calculator." http://www.micron.com/support/power-calc.

[59] Y. Lee, A. Waterman, R. Avizienis, H. Cook, C. Sun, V. Stojanovic, and K. Asanović, "A 45nm 1.3 GHz 16.7 double-precision GFLOPS/W RISC-V processor with vector accelerators," in *The 40th European Solid State Circuits Conference (ESSCIRC)*, 2014.

[60] C. Sun, M. T. Wade, Y. Lee, J. S. Orcutt, L. Alloatti, M. S. Georgas, A. S. Waterman, J. M. Shainline, R. R. Avizienis, S. Lin, *et al.*, "Single-chip microprocessor that communicates directly using light," *Nature*, vol. 528, no. 7583, pp. 534–538, 2015.

[61] "The ccbench micro-benchmark collection." https://github.com/ucb-bar/ccbench/wiki.

[62] "CoreMark EEMBC Benchmark." https://www.eembc.org/coremark/.

[63] "Cint2006." https://www.spec.org/cpu2006/CINT2006/.

[64] P. S. Li, A. M. Izraelevitz, and J. Bachrach, "Specification for the FIRRTL Language," Tech. Rep. UCB/EECS-2016-9, EECS Department, University of California, Berkeley, feb 2016.

[65] "Strober." http://www.strober.org, 2016.