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

Donggyu Kim, Adam Izraelevitz, Christopher Celio, Hokeun Kim, Brian Zimmer, Yunsup Lee, Jonathan Bachrach, Krste Asanović
Department of Electrical Engineering and Computer Sciences, University of California, Berkeley
{dgkim, adamiz, celio, hokeunkim, bmzimmer, yunsup, jrb, krste}@eecs.berkeley.edu

Abstract—This paper 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.

Index Terms—Design, Energy, Experimentation, FPGA, Hardware, Modeling, Performance, Power estimation, Statistical sampling

I. 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 microarchitectural 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 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%
II. 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 [20], [21], [22], [23], [24], [25]. 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 [26], [27], [28], [29]. Shafi et al. [26] validate an event-driven power model against the IBM PowerPC 405GP processor. Mesa-Martinez et al. [27] 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. [28] 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. [29] 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 [30], [31], [32], [33], [34]. Sunwoo et al. [30] generate power models from manually specified signals, which requires designers' intuition. This technique also requires additional manual efforts to instrument which can bottleneck emulation performance without careful partitioning. Atienza et al. [34] 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 [35], alternate at fixed intervals between detailed simulation (including a non-recorded warming stage) and fast functional simulation [13], [36], [37], [38]. 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 [39], [40], [17], [41], [42], [43]. Protobuf [39] implements a multi-core functional simulator on the FPGA. FAST [40] 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 [42] 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.

III. SAMPLE-BASED ENERGY SIMULATION

In this section, 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 III-A with parameters in Table I. Next, we describe how statistical sampling is applied to RTL energy simulation in Section III-B.
and plotted as a histogram (Figure 1). The distribution of these sample means (sampling distribution) has a variance $\text{Var}(\bar{x})$ (sampling variance) and a mean (sampling mean) that is equivalent to $\bar{X}$.

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

$$\text{Var}(\bar{x}) \sim \frac{s^2(n \div n)}{N_n} \quad (6)$$

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 $(1 \div \alpha)$ 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.

$$\bar{x} \pm z_{1-(\alpha/2)} \sqrt{\text{Var}(\bar{x})} \quad (7)$$

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.\(^3\)

**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 $\varepsilon$ represents the maximum relative difference allowed between the estimated parameter and the unknown true population parameter.

$$n \div \max \frac{\varepsilon^2}{z_{1-(\alpha/2)}^2 s^2} \geq 30 \quad (8)$$

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

**B. 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.
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 III-A). Since knowing the length of a full program execution is impossible a priori, we employ reservoir sampling [44] 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 $k$th 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 V-B, 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 [35], [13], [36], there is no state warming between timers.

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.

**IV. The Strober Framework**

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

### A. Chisel

Chisel [16] is a hardware construction language embedded in Scala [45] 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 [46], 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 intermediary representation, allowing for multiple optimization passes before the final RTL code is generated.
all components can be hosted on a single FPGA. In our case studies discussed in Section V-A, 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, 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 IV-B2). To complete FAME1 transforms, a global enable signal is connected by traversing all state elements in the Chisel backend.

2) State Snapshotting using Scan Chains: The sample-based energy simulation methodology described in Section III 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) to meet these requirements.

Figure 3 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 on-the-fly, so we must wrap each RAM with a scan chain first (Figure 4) and then add the RAM to a scan chain.
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\(^7\) 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 III-A, 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.

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\(^8\) to match nodes between RTL designs and gate-level netlists (Figure 5).

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.

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 Strobe’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 [53]. 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.

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.

C. **Replaying on Gate-level Simulation**

The FPGA RTL simulators generated by the Strobe framework provide cycle-exact performance estimates, but the replayable RTL snapshots must be simulated on a gate-level simulator to compute average power. Figure 5 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\(^4\) and place-and-route tool\(^5\) produce the gate-level netlist and the post-layout design, respectively. Gate-level simulation\(^6\), with very detailed timing, simulate the post-layout design to compute signal activities.

Replayable RTL snapshots are obtained from the Strobe platform-mapping transformation that automatically generates the correct interface for a specific FPGA platforms. This transformation (Figure 4) 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.
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 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.

D. 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. [55], we use Micron’s LPDDR2 SDRAM S4 [56] 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 [57].

E. 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 the gate-level simulations. Two-way out-of-order processor while $T_{ASIC}$ is around three or four hours. Also note that $T_{FPGA_{syn}} \geq T_{FPGA_{sim}}$ for real-world long-running applications.

To estimate $T_{FPGA_{sim}}$, 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 [44] ensures that the number of elements recorded during the simulation is roughly $2ln((N/L)/n))$ with the sample size n. The FPGA simulation time, $T_{FPGA_{sim}}$, is therefore:

$$T_{FPGA_{sim}} = T_{run} + T_{sample} \cdot N/LK_f + T_{rec} = 2ln((N/L)/n))$$

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 IV-C2). 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 Section III, each replay in the sample are independent and can be parallelized. Therefore, assuming $P$ instances of gate-level simulation, the total replay time is:

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

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, the FPGA simulation runs at 3.6 MHz, and the gate-level simulation runs at 12 Hz. In 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 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:

$$T_{FPGA_{syn}} = 3600 s$$

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

V. EVALUATION

A. Target Designs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Rocket</th>
<th>BOOM-1w</th>
<th>BOOM-2w</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch-width</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Issue-width</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Issue slots</td>
<td>-</td>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td>ROB size</td>
<td>-</td>
<td>24</td>
<td>32</td>
</tr>
<tr>
<td>Ld/St entries</td>
<td>8/8</td>
<td>8/8</td>
<td>8/8</td>
</tr>
<tr>
<td>Physical registers</td>
<td>32/int/32/32/32</td>
<td>100</td>
<td>110</td>
</tr>
<tr>
<td>L1 is and D is</td>
<td>16KiB/16KiB</td>
<td>16KiB/16KiB</td>
<td>16KiB/16KiB</td>
</tr>
<tr>
<td>DRAM latency</td>
<td>100 cycles</td>
<td>100 cycles</td>
<td>100 cycles</td>
</tr>
</tbody>
</table>

TABLE II: Processor Parameters

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.

B. Simulation Performance

<table>
<thead>
<tr>
<th>Simulation Cycles (10^9)</th>
<th>LinuxBoot</th>
<th>Coremark</th>
<th>gcc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record Counts</td>
<td>0.5</td>
<td>3.92</td>
<td>73.39</td>
</tr>
<tr>
<td>Simulation Time with Sampling (min)</td>
<td>12.88</td>
<td>32.80</td>
<td>344.00</td>
</tr>
<tr>
<td>Simulation Time without Sampling (min)</td>
<td>3.68</td>
<td>11.00</td>
<td>312.25</td>
</tr>
</tbody>
</table>

TABLE III: 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 III shows the performance evaluation of Strober with the two-way BOOM processor running long benchmarks showed in VI-A. 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 both can be synthesized at 50 MHz on the same ze706 FPGA.

C. DRAM Timing Model Validation

Fig. 7: 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.
Fig. 8: Comparison for the Theoretical Error Bounds with the Actual Errors

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Simulated Cycles</th>
<th>Replayed Cycles</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>vvadd</td>
<td>200521</td>
<td>30→128</td>
<td>1.92%</td>
</tr>
<tr>
<td>towers</td>
<td>410752</td>
<td>30→128</td>
<td>0.93%</td>
</tr>
<tr>
<td>dhrystone</td>
<td>396790</td>
<td>30→128</td>
<td>0.97%</td>
</tr>
<tr>
<td>qsort</td>
<td>187160</td>
<td>30→128</td>
<td>2.05%</td>
</tr>
<tr>
<td>spmv</td>
<td>927144</td>
<td>30→128</td>
<td>0.41%</td>
</tr>
<tr>
<td>dgemm</td>
<td>1833075</td>
<td>30→128</td>
<td>0.21%</td>
</tr>
</tbody>
</table>

TABLE IV: Simulated and Replayed Cycles for Each Benchmark on the Rocket Processor

D. Power Validation

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 8. We repeated this process five times for each benchmark.

VI. CASE STUDY

A. 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 [61]. 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[62] on Linux. For gcc, we execute the first 20B instructions (or 20%) of the SPECint reference input workload “gcc 166.in”.

B. Performance, Power, and Energy Analysis

Figure 9(a) 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.

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.
Fig. 9: Case Studies for the Strober Framework using the Rocket and BOOM Processors

VII. Case Study

turn-around time for evaluating 70 billion cycles on BOOM2w is approximately 7 hours for a complete evaluation. We believe this is fast enough to enable real-time feedback in the RTL design loop.
Fig. 10: 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.

...strated 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.

Strober is open-source and freely available [63]. The commercial CAD tools used in this paper are industry-standard, and widely available to academics through academic licensing programs.

ACKNOWLEDGEMENT

Research partially funded by DARPA Award Number HR0011-12-2-0016; the Center for Future Architecture Research, a member of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA; and ASPIRE Lab industrial sponsors and affiliates Intel, Google, HP, Huawei, LGE, Nokia, NVIDIA, Oracle, and Samsung.

REFERENCES


