Master's Thesis

AC/DSIM: FULL-SYSTEM ENERGY ESTIMATION WITH MODULAR SIMULATION

JONAS KAUFMANN

December 16, 2024

Advisor:
Dr. Antoine Kaufmann Operating Systems Group

Examiners:

Dr. Antoine Kaufmann	Operating Systems Group
Dr. Laurent Bindschaedler	Data Systems Group

Operating Systems Group
Max Planck Institute for Software Systems
Saarland University

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken,______________________ ______________________________
(Datum/Date) (Unterschrift/Signature)

ABSTRACT

Heterogeneous computer systems incorporate specialized hardware like FPGAs and ASICs for significant performance and energy efficiency improvements over CPU-only systems. However, meaningful evaluation and comparison of these systems requires assessing full-system performance and energy usage, which is often challenging due to the unavailability of physical prototypes. Full-system simulation tools offer an alternative by modeling system components and their interactions in detail, allowing researchers to estimate full-system performance. These tools often either only provide energy estimates for single components though, or abstract away a lot of detail, leading to low accuracies.

This thesis introduces AC/DSim, a modular framework for full-system energy and performance estimation, which extends SimBricks full-system simulations with modular energy estimation. We built AC/DSim to integrate detailed power models for components such as CPUs, caches, memory, hardware accelerators, and networks. By running complete software stacks in simulation, AC/DSim collects accurate workload information, enabling accurate power estimation throughout the workload's execution. The framework supports a modular combination of power models, allowing adaptation to various system designs and use cases.

We evaluate AC/DSim for a simple heterogeneous system featuring an ARM CPU and an FPGA-based deep learning accelerator. Preliminary results demonstrate AC/DSim 's feasibility to combine simulators and power models to estimate full-system performance and energy usage. While AC/DSim can efficiently combine existing power models, simulation speed varies significantly depending on the simulators used and their inherent overhead for logging the workload information power models require. Initial findings highlight modeling errors in the CPU simulator we use, leading to inaccurate performance and energy estimations. Nevertheless, AC/DSim successfully captures dynamic power trends, suggesting its potential for accurate system-wide evaluation once modeling limitations are addressed. This work lays the foundation for enabling comprehensive evaluation of heterogeneous systems, particularly those involving ASICs, where physical prototypes are unavailable.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

1 INTRODUCTION

We build increasingly heterogeneous systems that use specialized hardware such as FPGAs and ASICs to offload computations from the general-purpose processor. Key metrics we aim to improve while doing so are full-system performance, such as request throughput or latency, and full-system energy usage, for example, the energy consumed per request. Introducing specialized hardware can provide orders of magnitude improvements for both these metrics [19, 22].

However, evaluating such systems often either lacks one of these two metrics or does not take the full system into account. [36] Evaluating the hardware accelerator in isolation by reporting its peak performance and energy usage does not suffice. Moving data for the computation from host memory to the accelerator incurs latency overheads, which can quickly become a bottleneck for full-system performance. [11, 19, 38] Further, considering just the energy usage of the accelerator and comparing that to running the same computation on the CPU only yields an incomplete picture since the CPU still uses energy while waiting for the accelerator to finish its computation. Therefore, only when taking full-system performance and energy into account can we meaningfully compare different designs of heterogeneous systems [36].

In practice, measuring full-system performance and energy usage is hard because, depending on the system design, building a physical prototype, which we can measure as usual, is often infeasible. In some cases, we can physically prototype and measure systems using an FPGA for the hardware accelerator [9, 21, 23]. However, FPGAs are not representative of ASICs, which clock an order of magnitude higher and also feature orders of magnitude better energy efficiency. [22] Yet, FPGAs are usually the only option to evaluate as a full system [18, 43, 48]. When dealing with ASIC clock speeds, data movement bottlenecks also tend to show up, so using FPGAs to evaluate system designs that deploy ASICs is only convincing if the FPGA implementation is already competitive in terms of performance or energy efficiency.

Full-system simulation tools [24, 27, 33, 35] offer an alternative to evaluating physical prototypes by modeling each component and their interactions. For example, the SimBricks [24] modular full-system simulation framework provides full-system performance evaluation for small and large computer systems by combining and connecting different simulators for different components. While these can be used to evaluate the performance of the full system, they lack a means of evaluating its energy usage. Even though other tools exist that take both energy and performance into account [11, 29, 38], they often either target only the energy usage single components or abstract away too much detail with high-level modeling. Further, these tools are typically specialized for narrow use cases and, by construction, are not extensible to other devices, workloads, or larger systems.

This thesis aims to close this gap and presents a first step towards enabling meaningful evaluation of heterogeneous systems. We introduce AC/DSim (AC/DSim), a modular full-system energy estimation framework built on the SimBricks full-system simulation framework that combines modular simulation for performance results with modular power estimation for energy full-system energy usage.

The key idea in AC/DSim is to combine power models for components like CPUs and caches [10, 25, 40, 46, 47], memory [8, 14], hardware accelerators [7, 11, 31, 37, 38, 42, 44, 47], SSDs [30], and the network [13, 20, 29, 34] from prior work. Users can modularly choose the power models for each component that best suits their use case. These power models require accurate workload information for accurate estimates, which we collect in a SimBricks full-system simulation by running the full software stack, which includes the operating system, device drivers, and workload applications. The virtual testbed includes all hardware components, simulating interactions between them, for example, data movements when invoking the accelerator. During the simulation, we periodically sample workload information, which we use to invoke individual power models multiple times and therefore provide estimates of how a component's power draw changes dynamically throughout the workload's execution. Finally, to compute the full-system energy usage, we combine the power estimates during post-processing.

Our goal for this thesis is to demonstrate the feasibility of this idea, which we evaluate by estimating the energy usage and full-system performance for a simple heterogeneous system consisting of a small ARM CPU and an AXI-attached FPGA in the form of a system on a chip. On the FPGA, we deploy an open-source deep-learning hardware accelerator [4]. Even though we evaluate AC/DSim for an FPGA system, we ultimately target the evaluation of systems for which physical prototypes are unavailable, such as when designing ASICs. We argue that FPGA power models and the workload information they require are similar to ASICs, from which we claim that we can also support the evaluation of ASIC systems.

During our evaluation, we find that AC/DSim can indeed efficiently combine existing power models and simulators to provide full-system energy and power estimates. Depending on the sampling period the user chooses, we do so with negligible overhead on disk space required to store the workload information. Depending on the simulator, however, enabling the collection of the workload information can either have no impact on simulation speed or significantly slow it down, making it simulate up to twice as long. Our concrete simulation implementation suffers from a heavy slowdown of more than 1'000'000 times over real-time for gate-level simulation of the hardware accelerator even when not logging workload information, meaning we need to simulate for 1'000'000 s to forward virtual time by just 1 s. This is a fundamental limitation of the simulator we chose for the hardware accelerator but not for AC/DSim. We outline how to use alternative simulation approaches for the hardware accelerator, which are orders of magnitude faster but also possibly less accurate. Due to the slow simulations, we only provide preliminary results on the accuracy of performance and energy estimates using short workloads. For these workloads, we observe a significant modeling error in our CPU simulator, which causes errors of roughly $-40\,\%$ for estimated full-system performance, which is also reflected in the full-system energy estimates with errors of up to -27 %. The per-component power time series can accurately capture the dynamic evolution of power draw throughout the workload execution though, hinting that when fixing the modeling errors, AC/DSim can accurately estimate full-system energy and performance.

2 BACKGROUND

In this chapter, we provide the necessary background to follow this thesis. We start with the fundamental difference between power draw and energy usage, which are two key concepts throughout this thesis. Further, we discuss that the power draw of modern chips has two components: static and dynamic. The latter depends on the workload and accounts for the majority of total power draw, highlighting the need for high-quality workload information when aiming to accurately estimate power draw. We then introduce existing power models for different components, which we are going to use to implement our proposed framework for a simple heterogeneous system. For collecting the workload information that these require to accurately estimate dynamic power, our framework's later introduced design relies on full-system simulation. Therefore, we also present the necessary simulation fundamentals, especially regarding SimBricks, the full-system simulation framework we use.

2.1 POWER DRAW AND ENERGY USAGE

Power draw and energy usage are two important concepts we use throughout this thesis. While energy E measures how much electricity a computation uses, power P is the momentary rate of energy usage. Energy usage is, for example, an important consideration when designing systems that operate off a battery because it determines how long the battery will last [29]. It also controls the operating costs of a system [36]. In contrast, power draw is interesting when discussing peaks and valleys to appropriately size the cooling and power supply for a chip [17].

In the electrical setting, power is the product of Voltage V and Current I and expressed in Watts (W). Energy is the integral of power draw over time and as a result, its unit is Watts $*$ Seconds or Joule (J).

In practice, we cannot measure energy usage directly. Instead, we need to periodically sample the power draw of a physical component using a power sensor, which has a specific sampling frequency $F_{sampling}$ or temporal resolution. Since this yields measurements only for discrete time points, we can't compute energy using the integral. Instead, we can approximate actual energy usage, assuming that the power draw remains constant for one sampling period.

$$E \approx \sum_{0 \leq i < N} x_i * T_{sampling} \tag{2.1}$$

where

- N is the number of samples
- x_i is the i-th power measurement
- $T_{sampling}$ is the sampling period and equal to $\frac{1}{F_{Sampling}}$

In this thesis, whether we are measuring or estimating power draw, we always follow this formula to compute energy usage.

2.1.1 *Power Consumption of Modern Hardware*

Total power draw of modern hardware can be broken down into two parts, static and dynamic power [31] and both are significant [12]. Static power is the amount of power consumed when transistors are powered but do not switch. The most significant draw stems from transistor leakage current:

$$P_{static} \approx V_{dd} I_{leakage} \tag{2.2}$$

where

- V_{dd} is the supply voltage of the chip
- $I_{leakage}$ is the leakage current

$I_{leakage}$ mainly depends on the transistor fabrication technology and temperature [31].

In comparison, dynamic power is the power CMOS transistors consume while switching and can be expressed as

$$P_{dynamic} = \alpha C_L V_{dd}^2 F_{clock} \tag{2.3}$$

where

- α is the switching activity or average number of transistor switching per cycle
- C_L is the load capacitance
- is the clock frequency

For modern hardware, dynamic power draw is the most significant part of total power draw [12]. To estimate it accurately, we especially need to consider F_{clock} and α. More concretely, the key point of AC/DSim presented in Chapter 4 is to compute accurate switching activities for transistors throughout a chip, which is determined by the workload that uses the microchip.

2.2 PER-COMPONENT POWER MODELS

After discussing the fundamental factors determining static and dynamic power draw in Section 2.1.1, we now present power models for hardware accelerators, CPU, and CPU caches. These models estimate the total power draw of individual components. Our approach for AC/DSim relies on summing individual per-component estimates into a full-system power draw, which we can then use together with Equation 2.1 to compute full-system energy use. In this thesis, we consider three components: hardware accelerators, CPUs, and CPU caches. For hardware accelerators, we assume that their hardware source code is available. For CPUs, including their caches, however, this is usually not the case. We therefore turn towards black-box power models, which operate using proxy metrics instead of signal activities to estimate dynamic power.

2.2.1 *Hardware Accelerators Power Models*

We introduced equations to compute the power draw of transistors in Section 2.1.1. In practice, however, these are too low-level and quickly become unpractical as the size of the circuit increases [31]. Instead, we choose power models that operate at higher abstraction levels, concretely the level of gates, which are made up of multiple transistors. Every gate has one or multiple boolean in- and output signals. At this level, we approximate dynamic power by using boolean signal activity of the inputs instead of transistor switching activity. Although our approach works for ASICs as well, we focus on FPGA power estimation throughout this thesis since we don't have a physical open-source ASIC available that we can use to evaluate against. In this section, we therefore first discuss FPGA power models in detail and then show the similarities to ASIC power models.

For AMD and Intel *FPGAs*, their respective EDA suites, namely *AMD Vivado* and *Intel Quartus Prime Pro*, provide post place-and-route power analysis tools, which take files containing boolean circuit signal activities as input to provide more accurate power estimates [2, 17]. These tools are proprietary and their exact algorithms are unknown. In this thesis, we focus on Vivado since we have access to a physical AMD Xilinx FPGA for the evaluation of AC/DSim. The approach for Intel Quartus Prime Pro should be analogous though.

The power estimation uses the fully placed and routed design. In case the circuit signal activities are unknown, Vivado will estimate circuit signal activities, which is generally very inaccurate. For the hardware accelerator we use during evaluation, Vivado estimates 1,721 W, whereas the actual power draw when idle is 0.25 W. By feeding in accurate circuit signal activities, Vivado estimates much more accurate 0.49 W. So, power estimation without circuit signal activities is very inaccurate, especially because it cannot capture power draw varying over time as the utilization of the hardware accelerator changes, as in a real system.

To compute accurate signal activities, we must perform a gate-level simulation using Vivado's built-in simulator *xsim*. This requires stimulating the hardware accelerator's wires going into the design. To collect accurate signal activities, this stimulus has to match how the workload software running in the system would interact with the hardware. For this, we could use a Verilog testbench that models the workload, which is cumbersome to write and has the potential for modeling errors. During modeling, we must ensure we accurately capture the effects of software running on the CPU this hardware accelerator is connected to, i.e., operating system, driver, and applications. To easily collect this information without modeling errors, we use full-system simulation, which we introduce in Section 2.3.

Alternatively, we can also perform an RTL-level simulation, which is faster but incurs a loss of accuracy. The reason is that RTL is again a higher abstraction level than gates. Expressions in RTL don't directly represent gates. For example, during Synthesis, Vivado will take simple boolean operations, possibly merge them as an optimization, and express them via one or many LUTs (lookup tables), which are the actual *gates* the FPGA provides. This means that objects in the RTL hierarchy can be completely replaced during synthesis. The in- and outputs of modules typically remain intact though. Consequently, Vivado will only be able to match the activities for a fraction of the signals. For example, for a simple JPEG decoder [16], we go from 98 % of signal activities matched when using gate-level simulation to just 20 % for RTL-level simulation.

For unmatched signals, Vivado runs its algorithm to estimate signal activities. According to their documentation, Vivado still takes the matched signals into account to estimate the missing activities [2]. So the RTL estimation is still better than no workload activities. In this thesis, we will focus on gate-level simulation since that enables the most accurate power model for an FPGA.

Even though we focus on FPGA estimation in this thesis, the approach for ASICs is the same from a user's perspective. All of these models require signal activities for accurate results and perform estimation using signal activities either collected during a gate- or RTL-level simulation. Similarly to Vivado, Synposys Prime Power [42] and Cadence Genus [7] both use information from gate-level or RTL-level simulation to provide signal activities [31, 42]. Meanwhile, OpenSTA uses only RTL-level simulation results [44].

2.2.2 *CPU and CPU Caches Power Model*

To estimate the power of CPU and CPU caches, we cannot rely on built-in power estimation tools of EDA suites since we typically cannot access the hardware source code for today's CPUs from AMD, Intel, and ARM. Therefore, we do not know the signals within the CPU or its caches and have no way of collecting signal activities in order to estimate dynamic power at gate-level using one of the power models presented in the prior section. Instead, we switch to black-box power models that rely on higher-level metrics, namely performance monitoring counters, for estimating dynamic power

Previous work explored building linear regression-based power models that estimate the power draw of CPU and CPU caches by using the CPU's *performance monitoring counters (PMCs)* as input [10, 47]. These count, for example, the number of retired instructions, cache misses or hits, and non-halted cycles. They are quite accurate with the authors reporting a 1.5 % and 3.8 % error across different workloads, respectively. The approach is to build these directly from the physical target CPU by executing a wide selection of open-source benchmarks and periodically logging PMCs, as well as power consumption. The resulting linear regression models take the CPU's PMCs as in- and output an estimate for the power draw. For AC/DSim, we simulate the CPU though, and the simulator we use, namely *gem5* [6, 26], does not provide the performance monitoring counters that these models require. Instead, we rely on GemStone [46], which is an extension of [47]. GemStone uses the power models built using the approach presented in [47] and finds gem5 performance statistics that correlate well with the CPU's PMCs, which it then uses instead of the PMCs as the power model's inputs.

Again, GemStone builds the linear regression power model using PMC and power measurements from the actual CPU that we want to model. To estimate power, however, we run the same workload in gem5 and use gem5 performance statistics in place of the selected PMCs as input to the linear regression models. The GemStone authors found some of gem5's performance statistics to be inaccurate and eliminated their corresponding PMCs from the PMC selection process while building the power models. Overall, their final linear regression model for an ARM Cortex A15 achieves a 10 % *mean average percentage error (MAPE)* for the power estimation when compared to actual CPU power measurements across a wide range of workloads.

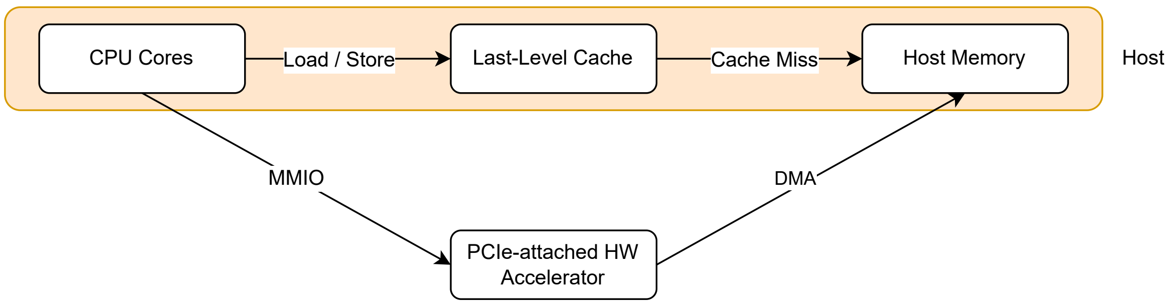


Figure 2.1: The components of a simple heterogeneous system. The CPU can offload computations to a PCIe-attached accelerator. During execution, components send requests to other components in the direction of the arrows and receive data responses in the opposite direction. We call CPU, its caches, and the host memory *host*.

2.3 FULL-SYSTEM SIMULATION

After introducing the power models we use in this thesis, we now present how we feed them with accurate workload information for dynamic power estimation by simulating full systems using SimBricks [24], which efficiently connects and synchronizes existing simulators for individual components into a full-system simulation. In this virtual system representation, we can boot Linux and run the actual software, including device drivers and workload application, and log all information the power models require.

2.3.1 *Simple HW Accelerator System: Important Components and their Interactions*

We first give a rough overview of important system components and interactions between them using a simple heterogeneous system as an example, before introducing how we simulate such systems. We show this simple system in Figure 2.1. It contains a CPU together with its caches, host memory, and a HW accelerator.

During regular software execution that doesn't involve the hardware accelerator, the CPU issues load or store requests to its cache hierarchy. If these requests miss in the caches, the *last-level cache (LLC)* will forward them to the *host memory*. If, however, workloads involve computations offloaded to the accelerator, the driver is at some point invoked by the workload and issues *Memory-Mapped I/O (MMIO)* requests to the accelerator to read or write its control registers and thereby program it. Once the driver sends a start signal via MMIO, the accelerator begins execution and sends read and write requests to the host memory to retrieve input data or write back results. For this, it uses a mechanism called *direct memory access (DMA)*.

An alternative design used in today's heterogeneous systems since caches are no longer a scarce resource, is hooking up the accelerator to the LLC. This helps with throughput and latency, especially for non-sequential memory accesses. Another upside is that this means the driver need not flush the CPU caches when invoking the accelerator after having written input data to DMA memory regions. We do not consider DDIO for our prototype implementation of AC/DSim for a concrete heterogeneous system we present in Section 4.1 but want to stress that this is not a limitation of our framework.

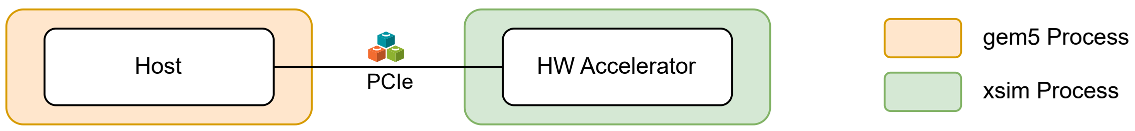


Figure 2.2: The instantiation of the simple heterogenous system from Figure 2.1 in the full-system simulation framework SimBricks (). SimBricks connects the two simulator processes by forwarding requests and responses via shared memory queues and synchronizing the two simulators.

2.3.2 *Assembling Full-System Simulations with SimBricks*

To simulate a simple heterogeneous system such as the one shown in Figure 2.1, we rely on SimBricks [24] as it allows us to modularly combine simulators into a virtual testbed containing all hardware components and which can run the whole software stack including operating system, drivers, and workload applications.

Concretely, SimBricks connects battle-tested simulators for individual system components at natural boundaries like PCIe and Ethernet. In our case, the accelerator is attached to the host via PCIe, which SimBricks models at the transaction level. In the SimBricks abstraction, MMIO and DMA are simple read and write requests that can be sent in both directions.

SimBricks runs simulators as individual processes and exchanges messages between them via shared-memory queues. Figure 2.2 shows the instantiation of the simple heterogeneous system presented in the prior section in a SimBricks full-system simulation. For read requests from simulator $A \to B$, for example, SimBricks sends a message from $A \to B$. Once B fetched the data, SimBricks, sends a message containing the data back from $B \to A$.

SimBricks connects different simulator types, for example, event-driven like our host simulator, and cycle-accurate, like the RTL- and gate-level simulators we use for the hardware accelerator. While doing so, every simulator retains its internal notion of a clock. For meaningful performance measurements, SimBricks synchronizes these clocks of individual simulators. SimBricks' synchronization mechanism is in-band by sending special messages in the shared memory queues. It also exploits link latencies to avoid costly lock-step synchronization while remaining fully accurate. This is very efficient. The authors of SimBricks authors showed that the synchronization overhead over the simulators' inherent already slow simulation speed is negligible. Still, the slowest simulator dictates the simulation speed of the full system, which is going to be an important consideration during our evaluation in Chapter 5.

SimBricks is straightforward to scale. To simulate larger systems like the one shown in Figure 2.3, which involve multiple hosts, and a network that connects them, SimBricks simply runs more simulator processes. Thanks to SimBricks' synchronization slack, the individual processes can all simulate in parallel, given enough CPU cores. Overall the slowdown when simulating a larger system is therefore negligible. [24]

For more technical detail on how SimBricks connects component simulators, have a look at the SimBricks architectural overview in its documentation [28]. Next, we offer more detail on the simulators we use to simulate hosts and the hardware accelerator, as well as the workload information these provide.

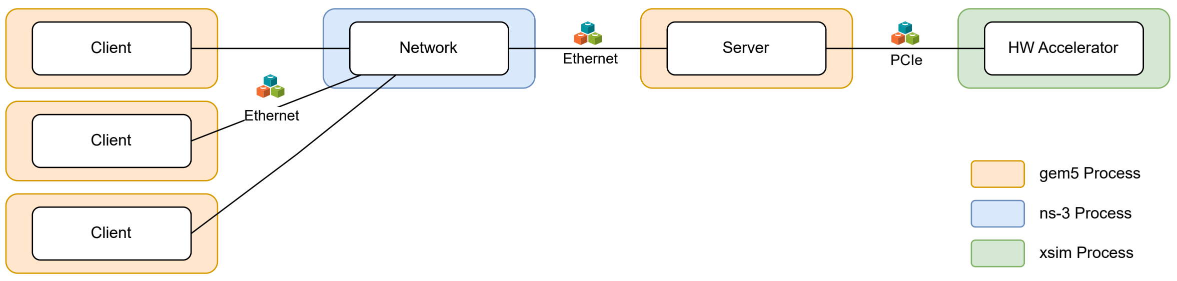


Figure 2.3: A larger heterogeneous system that involves multiple client hosts that send requests to a server host over the network. For faster and more efficient computations, this server uses a PCIe-attached accelerator. An application of such a system is mobile clients requesting machine learning inference from a backend server. To simulate such a system, we run individual simulator processes for every component shown in this figure and use SimBricks to connect them via natural boundary protocols like Ethernet and PCIe.

2.3.3 *Host Simulation*

We use gem5 [6, 26] to simulate the *host*, which spans CPU, its caches, host memory, and disk drives. This is the most accurate host simulator that SimBricks already supports and is widely used in research and industry as it is open-source and allows simulating x86, ARM, and RISC-V CPUs together with cache hierarchies and host memory. It also models host-level interconnects like the memory bus, to which all components attach that need to access the memory. All this is highly configurable, and gem5 comes with in- and out-of-order models for the CPU to allow matching arbitrary platforms. On the flip side, all this configurability also introduces risk for the user to introduce modeling errors, which is something that is going to come up during the evaluation in Chapter 5 again.

gem5 supports booting unmodified Linux off a provided disk image and therefore enables us to run the complete software stack for our workloads, including the driver. For accurate performance measurements with gem5, we must correctly specify parameters for CPU pipeline, cache, interconnects, and the memory.

To collect accurate workload information, we instruct gem5 to periodically dump execution statistics. We show an excerpt of such a dump in Listing 2.1. Aside from the current virtual timestamp (`simSeconds`) and the number of CPU cycles executed (`system.cpu_cluster.cpus0.numCycles`), this dump contains very detailed statistics, for example, the number of integer instructions decoded in the fetch2 pipeline stage (`system.cpu_cluster.cpus0.fetch2.intInstructions`) or the number of accesses to the L1 DCache (`system.cpu_cluster.cpus0.dcache.overallAccesses::total`). We use a few of the execution statistics gem5 provides as input to our CPU power model. More information on that in Chapter 4.

2.3.4 *HW Accelerator Simulation and Power Estimation*

For the simulation of the hardware accelerator, we rely on Vivado's [1] built-in RTL- and gate-level simulator called xsim. For power estimation, we also rely on the Vivado toolchain by using its built-in power estimation tooling [2]. We briefly introduce both in this section.

Concretely, we use xsim to collect signal switching activities at gate-level, allowing us to invoke Vivado's most accurate power estimation model, which runs after place-and-route [2].

```
simSeconds  0.101460    # Number of seconds simulated (Second)
[...]
system.cpu_cluster.clk_domain.clock 833 # Clock period in ticks (Tick)
[...]
system.cpu_cluster.cpus0.numCycles  1703943 # Number of cpu cycles simulated (Cycle)
[...]
system.cpu_cluster.cpus0.commitStats0.committedInstType::total  894948  # Class of
    committed instruction. (Count)
[...]
system.cpu_cluster.cpus0.fetch2.intInstructions 332326  # Number of integer
    instructions successfully decoded (Count)
[...]
system.cpu_cluster.cpus0.dcache.overallAccesses::total  248088  # number of overall (
    read+write) accesses (Count)
```

Listing 2.1: An excerpt of a gem5 stats.txt file showing various statistics collected during workload execution. These statistics are very detailed, containing information about pipeline stages, cache accesses, etc.

For this, we have two key requirements. First, the hardware accelerator's RTL code needs to be available. Second, it needs to be fully synthesizable without timing violations for all clock speeds we want to evaluate. Timing violations can lead to errors during computation when deploying the synthesized design on a physical FPGA. Concretely, we use the so-called post-synthesis functional simulation for xsim. Post-synthesis means that xsim uses a Verilog Netlist produced during Vivado's synthesis process, in which RTL constructs are replaced with logic resources available on the FPGA or, in other words, the analogous concept of gates on an FPGA. It is also possible to run the gate-level simulation place and route, which is called post-implementation functional simulation. We verified that this does not produce a number of signals matched when invoking the power model though. In fact, when using post-implementation functional simulation, we run into problems due to optimizations. The reason is that we do not attach to the synthesized design at the top-level, which we explain next.

To deploy a hardware accelerator to an FPGA, we need to put additional IP blocks around it. In the case of SoC-based AMD Xilinx FPGAs, the top-level IP block represents the rest of the SoC, which exposes connections for MMIO and DMA to the hardware accelerator. It does not provide unconnected wires that we can use to drive these though. For AMD Xilinx PCIe-based FPGA cards, the highest level IP block is a PCIe endpoint IP, which has wires to connect to PCIe sticking out. We also do not want to drive PCIe, so instead, for simulation of the accelerator, we a few levels into the topology, allowing us to interact with the accelerators and its wires sticking out directly. This does not work with post-implementation functional simulation though as some signals are short-circuited during optimization to reduce routing resources required. If we do not attach at the top-level signals, which will remain intact, this leads to broken behavior of the hardware accelerator.

We can dynamically execute commands in xsim to write signal activities for the whole testbench to a file using the specifically optimized SAIF format. For our implementation in Chapter 4, we rely on this feature to dynamically execute commands to sample workload information. SAIF, in contrast to VCD waveform files, does not log the value of each signal for every clock cycle but contains exactly 5 counters per signal. The three most important include the amount of time the signal had a value of 0, the amount of time the signal had a value of 1, and the number of signal transitions.

3 DESIGN

After providing the necessary background in power, energy, hardware power models, and full-system simulation in Chapter 2, we now present the design of our full-system power and energy estimation framework AC/DSim (AC/DSim). We focus on how we provide accurate workload information to power models using full-system simulation, which means users don't have to model workloads themselves. Our approach is modular, so given *suitable* power-model-simulator combinations, users can plug in power models that best suit their use case. In Chapter 4, we then discuss a concrete instantiation of our framework to estimate energy and power for a physical system.

3.1 DESIGN GOALS

Before diving into our framework's design, we present the goals we aim to achieve.

FULL-SYSTEM. We aim to estimate the energy usage of the whole system, considering all significantly energy-using components. Aside from hardware components, this includes software like kernel, drivers, and applications running in the system.

MODULAR. We want to make it easy for the user to swap out power models for system components or include additional models. This also includes the simulators we use to provide workload information.

ACCURATE. Full-system and individual hardware components' energy and power estimates should be accurate compared to the physical system to be reliably used to explore system- and component-level design choices and their energy performance trade-off.

SCALABLE. We aim to enable energy and power estimation of large-scale systems comprising multiple hosts with complex network topologies and many hardware accelerators.

3.2 HIGH-LEVEL OVERVIEW

With the goals defined, we now provide a high-level overview of the design of *AC/DSim*, which we also sketch in Figure 3.1. The end goal of applying our framework is to produce a full-system power time series that we can use together with Equation 2.1 to compute a full-system energy estimate. For this, we combine multiple power models from prior work to get power estimates for a subset of the system's components each. Then, during post-processing, we sum these individual estimates into a full-system estimate.

Power models from prior work can span one component, for example, the hardware accelerator [2, 7, 42, 44], or multiple like CPU cores and caches [46, 47], or even the complete mainboard with CPU, memory, caches, etc. [10]. For accurate individual power estimates,

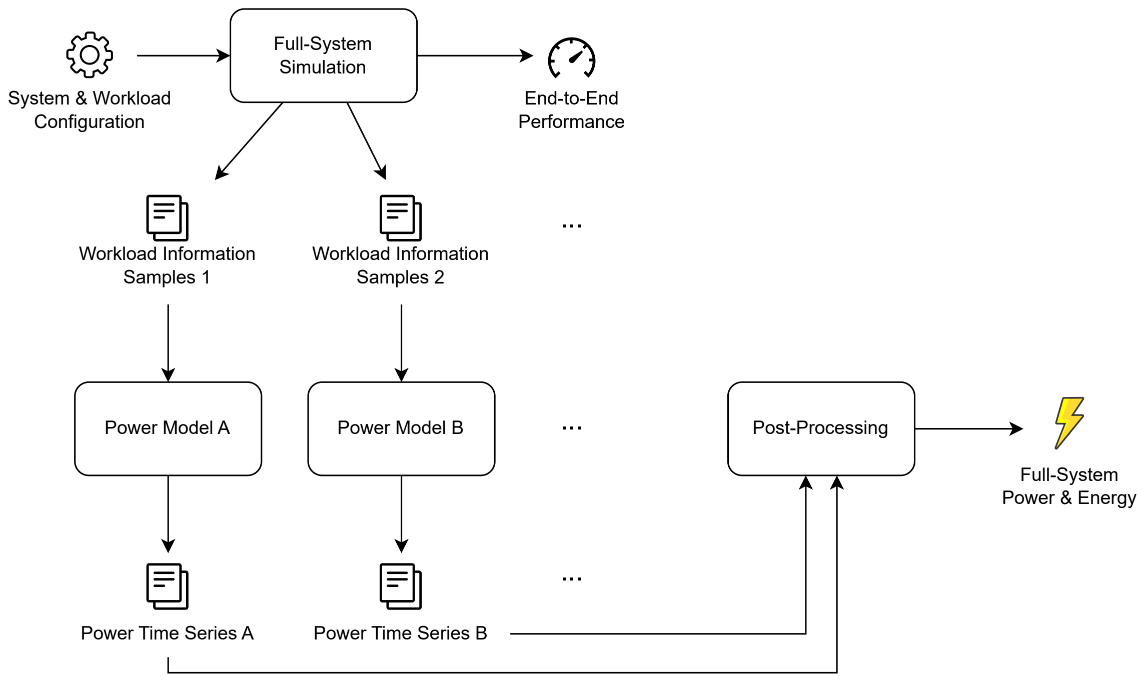


Figure 3.1: High-level overview of our framework's steps to compute full-system power and energy estimates. First, we run a full-system simulation that collects workload information samples, which we then feed into power models that span one or more system components to compute power time series. Finally, during post-processing, we sum the individual time series into a full-system power time series and full-system energy estimate.

we must consider dynamic power (see Section 2.1.1), which means we must supply the power models with workload information. We collect this workload information using a SimBricks [24] full-system simulation. This simulates all the system's hardware components and the interactions between them. Further, inside this *virtual testbed*, we run the complete and unmodified software stack, including Linux, drivers, and applications. Consequently, users don't need to model any part of the workload to extract workload information.

We periodically sample workload information from the simulators used for the full-system simulation to retrieve a power time series. After the full-system simulation is completed, we estimate power by invoking the per-component power models and feeding them with the collected workload information samples. We use exactly one workload information sample to compute one power estimation sample. This yields a power time series per power model. Ultimately, we sum all individual time series into one full-system power time series. Notably, individual time series remain available to enable inspection of how power is divided among components. Finally, we can easily compute full-system energy usage from the individual or full-system power time series using Equation 2.1. In this chapter, we do not discuss the details of invoking the individual power models and producing power time series since this depends on the combination of the concrete power model and simulator. Instead, we refer to Chapter 4.

3.3 DESIGN ASSUMPTIONS

To be able to apply AC/DSim for estimating the power draw and energy usage of a heterogeneous system, we need to fulfill the following assumptions.

1. *Compatible power model and simulator combinations.* We supply power models with workload information from a SimBricks full-system simulation. For this, we require that for every power model, we can find a combination of simulators used in the full-system simulation, where the simulators supply all workload information expected as input.

2. *Power models only use workload information independent of the sampling window length.* This is a pre-requisite to sample workload information and invoke the power models multiple times to compute a time series. Whether the workload information was collected for a larger or shorter sampling window must not matter. The workload information should, therefore, be expressed in rates, relative numbers like percentages, etc.

3. *Simulators record workload information independent of the sampling window length.* This goes hand-in-hand with the previous assumption. Often, workload information dependent on sampling window length can be pre-processed before invoking the power models to remove the dependency. We discuss this further in Section 3.5.

4. *Power models are disjoint.* We combine individual estimates from power models into full-system estimates by summing. For this, we require that power models don't overlap in the hardware components they cover.

5. *Simulation and power models have been validated against a physical testbed.* In general, simulation is a model of a physical system and is therefore prone to modeling errors. The same goes for the power models. We aim to show our framework's feasibility and therefore regard tuning simulation and power models to high accuracy as orthogonal work.

3.4 COLLECTING WORKLOAD INFORMATION IN A FULL-SYSTEM SIMULATION

We collect the workload information necessary to accurately estimate dynamic power by simulating the complete system, including all hardware and software components, in a SimBricks full-system simulation. As input to the full-system simulation, the user provides the system and workload configuration in the form of Python scripts. The *system configuration* specifies the hardware components, how they are connected, their configuration parameters, and which simulators to use for each component. Meanwhile, the workload configuration is concerned with the software side, i.e., which disk image to boot and which shell commands to run.

A nice property of simulation is that, although collecting detailed information may slow down the simulation, the behavior of the simulated system remains unaffected. Aside from this, the workload information existing simulators output is very detailed, even surpassing what is possible to observe in a physical system. Cycle-accurate gate-level simulators, for example, allow logging all signals in a design down to the level of individual logic resources or gates (see Section 2.3.4).

Our framework inherits the modularity of SimBricks. Either SimBricks already supports simulators that provide the necessary workload information for the power models used, or we can add additional simulators to SimBricks that do. This requires writing a simple adapter that translates the simulator's API into a SimBricks boundary protocol like PCIe, Ethernet, etc. This is out-of-scope for this thesis, but for more information, refer to *SimBricks Developer Guide - Architectural Overview* [28]. For power models we do not consider in this thesis, we are optimistic that they can be integrated into our framework since they only depend on a simulator that can provide the necessary workload information.

3.5 SAMPLING POWER ESTIMATES

The central technique used in our framework's design is the sampling of workload information at simulators. Workloads vary over time, and so does their usage of hardware resources and, thereby, power draw. Similarly to periodic measurements with a power sensor, we invoke the power models multiple times to provide us periodic estimates with respect to virtual time.

The user chooses the sampling frequency $F_{sampling} = \frac{1}{T_{sampling}}$. Assuming we want to compute a power sample for timestamp t, we use the workload information from the sampling window $(t - T_{sampling}, t]$. To collect the workload information samples, we instruct the simulators used in the full-system simulation to log workload information every sampling period $T_sampling$. This is where our design assumptions 2. and 3. (see Section 3.3) come in since for this sampling of workload information to work, we need that power models only use workload information that is independent of the sampling window length, like rates or percentages. Additionally, simulators also need to produce workload information that is independent of the sampling window length.

However, given unsuitable workload information produced by a simulator, we can often pre-process it before feeding it into the power model to make it adhere to assumption 3.. For example, as we can see in a gem5 workload information dump in Listing 2.1, the simulator provides only a cumulative counter of executed CPU instructions since the start of the simulation, which violates assumption 3.. However, we can transform it into a suitable workload information metric by using the virtual timestamp, also part of this dump, thereby getting rid of the dependency on time. By comparing the timestamp of the previous dump $tn - 1$ with the one of the current dump t_n, and using the number of CPU instructions from the previous dump x_{n-1} and from the current dump x_n, we can calculate the rate of CPU instructions executed per second for the current sampling window using $\dot{x} = \frac{x_n - x_{n-1}}{t_n - t_{n-1}}$.

4 IMPLEMENTATION

After discussing the design of AC/DSim, we now describe how we implement full-system energy estimation for a simple heterogeneous system, which we also use for evaluation. We show how we collect the workload information samples from the selected simulators. Further, we highlight the implementation challenges we faced and how we partially solved these, such as very slow gate-level simulation. We conclude the chapter by showing how we combine the individual power models' time series into a full-system time series, taking into account that the timestamps of power estimation samples do not necessarily align.

4.1 OUR FPGA SOC BASELINE SYSTEM

We now introduce the simple heterogeneous system, for which we estimate energy usage using our framework. This system closely follows the one shown in Figure 2.1 and is based on a physical FPGA SoC board, which we use to evaluate accuracy in Chapter 5.

Unfortunately, we could not access any open-source and manufactured ASICs that could serve as a physically measured baseline while prototyping our framework. For this reason, we decided to target FPGAs, which, as discussed in Section 2.2.1, have similar power models to ASICs in the sense that both are integrated into EDA toolchains and both rely on signal activities as workload information for their power models. Concretely, we use an Avnet Ultra96-V2[1]. This SoC features four ARM Cortex-A53 in-order cores, a rather small FPGA with 154K system logic cells, and 2 GB of LPDDR4 memory. The limited feature set makes it a good fit to prototype our framework though. The SoC is relatively simple and heterogeneous, which means we can already test combining two power models, concretely, one for CPU and caches, and a second one for the FPGA. On the FPGA, we deploy an open-source hardware accelerator, which makes the system heterogeneous.

The SoC's board features independent power sensors for the FPGA, CPU, and host memory. However, their sampling frequency is very limited. We empirically determined that we can reliably sample with sampling periods as short as 80 ms. We do so by reading from Linux character device files, which the driver exposes. We can actually read these more often, but this will not provide a higher temporal resolution than roughly 80 ms. In fact, when sampling more often, the measurements seemed to lose temporal resolution and, sometimes, did not output values that corresponded to the workload we ran.

The accuracy of the power sensors is limited, although fine for our use case. The values in the Linux character device files use a fixed-point representation with 5 bits of precision after the decimal point With this, we get an accuracy up to $2^{-5} = 0{,}03125\,W$

In terms of high-level components and how they are connected, this FPGA SoC looks almost like our simple heterogeneous example system from Chapter 2, shown in Figure 2.1

1 https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/ultra96-v2/ (Retrieved Dec 1, 2024)

The only difference is that on our physical system, the FPGA is connected via an *Advanced eXtensible Interface (AXI)* [5] instead of the PCIe bus to the host though.

At the transactional level of the PCIe protocol, there is no fundamental difference. The CPU still uses MMIO to read and write control register of the accelerator deployed on the FPGA And for accessing host memory, the accelerator also uses DMA, just like in the example system. SimBricks does not support the AXI protocol as a boundary between simulators yet. Due to the high-level similarities, we can instead just represent this AXI bus as a PCIe bus in simulation More on that in the next section, where we discuss how we represent the SoC in a SimBricks full-system simulation

4.2 SIMULATING THE FPGA SOC BASELINE SYSTEM

Our framework relies on SimBricks full-system simulation to collect the workload information necessary when invoking per-component power models. We introduced SimBricks in Chapter 2 in Section 2.3.

For the simulation of the host, which spans CPU, caches, host memory, and disk drives, we rely on gem5 (see Section 2.3.3 Meanwhile, we simulate the FPGA using Vivado xsim (see Section 2.3.4). Figure 4.1 provides an overview of all pieces involved in the full-system simulation. We now discuss these and highlight differences between the FPGA SoC board and the virtual testbed provided by the full-system simulation.

To simulate our FPGA SoC baseline system, we needed to make a few additions to SimBricks. So far, SimBricks has not been set up to run ARM simulations. This is not a fundamental limitation since gem5 can simulate ARM CPUs. Mainly, SimBricks did not support building ARM disk images, and there was no gem5 ARM configuration script that integrated with SimBricks. We added both and will upstream our additions. For now, we publish our additions to SimBricks[2] and gem5[3] on GitHub.

We based our gen5 configuration script on existing ARM example scripts and the existing SimBricks configuration script for simulating x86 hosts. For the CPU model, we selected gem5's HPI (High-Performance In-order) model, which is tuned to be representative of modern in-order Armv8-A implementations. We further verified that cache sizes and their associativity match the ARM Cortex-A53 of our physical FPGA SoC. Unfortunately, there was no pre-defined memory configuration that matched our LPDDR4 configuration, so we just went with the same configuration that the ARM example scripts also use (DDR4_2400_4x16). We decided against further steps to make the gem5 configuration closely match our physical host since our goal for this thesis is to show the feasibility of AC/DSim. We therefore regard tuning the simulation and power models to high accuracies as orthogonal, which we also reflect in the design assumptions presented in Section 3.3.

To simulate the hardware accelerator deployed to the FPGA, we rely on Vivado's integrated RTL- and gate-level simulator xsim (see Section 2.3.4). Even though SimBricks supports RTL-level simulation using Verilator [39], it does not integrate xsim yet.

Open-source hardware accelerators often expose top-level AXI [5] ports [4, 16] for interfacing with other hardware or system components. Therefore, we developed reusable AXI adapters that connect to these ports and translate AXI off the wire into the SimBricks PCIe

2 https://github.com/jonas-kaufmann/simbricks/tree/j-ma
3 https://github.com/jonas-kaufmann/gem5/tree/j-ma

boundary protocol and vice versa. For example, we read AXI DMA requests issued on the accelerator's top-level ports and turn them into SimBricks PCIe read or write requests. For the translation, we need to call C/C++ functions from Verilog, pass the values of Verilog signals to C/C++, and return values from C/C++ we want to apply in Verilog. This is possible via the SystemVerilog *Direct Programming Interface (DPI)* extension.

All this is completely transparent to the host simulator, meaning we don't have to change anything there. Our AXI to PCIe adapters don't handle synchronization between simulators though, which is required for the accurate simulation of interactions between individual simulators and the components they model. We implement this synchronization ad-hoc per accelerator in 40 lines of code total across Verilog and C/C++. We publish the Vivado integration on Gitlab[4].

SimBricks allows us to set a constant latency at simulator boundaries per direction independently. We empirically determined the AXI latency of the FPGA SoC to be roughly 250 ns one-way and tuned the SimBricks PCIe latency to reflect this Further, our AXI to PCIe adapters can accept multiple DMA requests in parallel, which the accelerators we consider do in fact make use of [4, 16] to hide host memory access latencies. Again, we empirically determined that our FPGA SoC can manage up to 16 pending AXI DMA requests and enforce the same limit in our AXI to PCIe adapters.

Software-wise, the physical FPGA SoC board runs PYNQ[5], which is an AMD Xilinx project aimed to make it easier to use their heterogeneous SoC platforms. PYNQ is based on Ubuntu 22.04. We deliberately don't run the same software in simulation as, for example, after boot, PYNQ starts a Jupyter Notebook server among other things, which means we would waste time simulating software we do not need. Instead, we build our own disk image based on an Ubuntu 24.04 minimal cloud image. Unfortunately, Ubuntu does not offer an ARM64 minimal cloud image for Ubuntu 22.04, which is why we had to use a newer version Although the versions of installed packages are different, we don't expect much influence from this on power draw or performance because, other than that, we run the exact same workload software in simulation that we also deploy to the FPGA.

For the hardware accelerator driver, we use slightly different implementations for performing MMIO in simulation and the FPGA SoC board. In the simulation, we use *Virtual Function I/O (VFIO)*[6], whereas, on the physical board, we just write to a physical address directly via the `/dev/mem` character device file. We expect the effect on accelerator invocation latency to be minimal since the control registers are only used for one-shot configuring the accelerator before invocation and while polling for its completion

4.2.1 *Speeding Up Gate-Level Simulation*

While trying out the first simulations, we ran into the problem that gate-level simulation is practically too slow to support simulating a workload only running for a second of virtual time. We observed a greater than 1'000'000 times slowdown over real-time for the full-system simulation, when using gate-level hardware simulation. This means having

4 https://gitlab.mpi-sws.org/jkaufman/jpeg-decoder-vivado-files
5 http://www.pynq.io/ (Retrieved Dec 1, 2024)
6 https://docs.kernel.org/driver-api/vfio.html (Retrieved Dec 1, 2024)

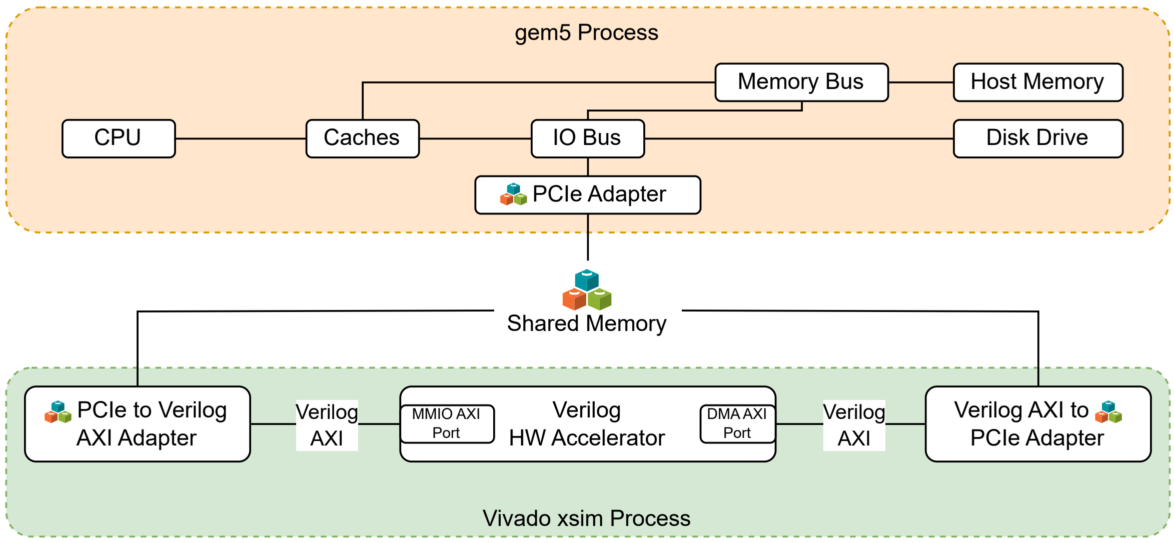


Figure 4.1: The representation of the baseline FPGA SoC board in a SimBricks full-system simulation. We use gem5 to simulate the host and Vivado xsim to simulate the FPGA. To connect the hardware accelerator deployed on the FPGA to the host, we use adapters that use SystemVerilog DPI to translate AXI transaction off the wire into SimBricks PCIe transactions and vice-versa. On the gem5 side, SimBricks provides an adapter that acts as a simulator-native PCIe device and exchanges SimBricks PCIe boundary protocol messages with the FPGA-side adapters through shared memory queues.

to simulate for 12 days for just a second of virtual time and in reality, the slowdown we observed was even higher.

During our experiments, we observed that the hardware accelerator is often only active for a fraction of the time, concretely up to 250 ms for the workloads we look at during our evaluation (Chapter 5). We can therefore optimize simulation speed by only simulating the hardware accelerator when it is actually in use. For this, we add a *pseudo-synchronization* mode to the hardware accelerator simulator. When pseudo-synchronization is active, the simulator still advances the virtual timestamp that it exposes through the SimBricks mechanism to other simulators and still synchronizes with connected simulators, but it does not evaluate the hardware accelerator. So even though it looks like the hardware accelerator is advancing time to the outside, the simulator itself is paused. This is a non-intrusive addition, which is fully transparent to other simulators.

To toggle the pseudo-synchronization mode, we need knowledge about when the accelerator is active. Since this is hard to predict before running the full-system simulation, we expose this control inside the simulation itself. gem5 has a similar mechanism, where a read or write to special physical addresses on the host controls, for example, the periodic logging of workload information. Here, we must send a similar signal to the hardware accelerator simulator through the SimBricks PCIe channel. The easiest way to do so is by exposing another PCIe base address register (BAR) just for simulation control. The hardware accelerator's driver is the best place where to make use of this simulation control since it is invoked when the workload requests accelerator service since it is the closest piece of software to the hardware accelerator that has knowledge of its semantics. So we modify the driver to disable pseudo-synchronization just before starting the accelerator. And after the accelerator finishes, the driver enables pseudo-synchronization again.

4.3 ESTIMATING ENERGY OF THE FPGA SOC BASELINE SYSTEM

To estimate the energy of our FPGA SoC baseline system, we mix two power models for CPU / caches and the hardware accelerator. These two components consume the most significant amount of energy when measuring our physical system.

4.3.1 *FPGA Power Estimation*

For estimating the power consumption of the FPGA, we use power estimation built into AMD Vivado. We briefly introduced the necessary background for this in Section 2.2.1 and Section 2.3.4.

This type of power estimation is convenient to use. We feed our hardware accelerator's RTL code into the synthesis tool (Vivado) and run it until place-and-route. This allows us to estimate power using the placed and routed design, which requires accurate signal activities collected in the gate-level simulation for accurate estimates.

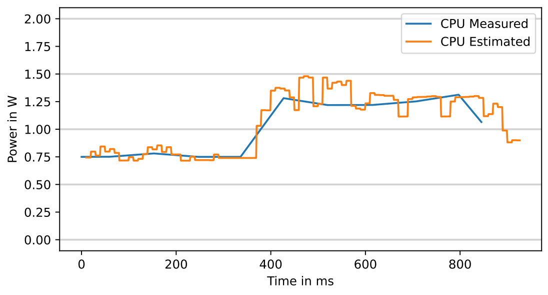
To collect these signal activities, we instruct Vivado xsim to write SAIF files, which are optimized to capture these. To sample signal activities, we use a TCL script that runs the simulation for one sampling period, dumps the SAIF file, resets the switching activity counters for all signals, and repeats this process.

The sampling period controls a trade-off between the amount of data logged and accuracy. Smaller sampling periods (higher sampling frequencies) are better capture fine-grained dynamic differences in power draw since the workload statistics represent an average over the sampling window. However, longer sampling periods (lower sampling frequencies) log fewer samples and, therefore, require less disk space. Further, since the power model is applied after the simulation, less samples also means that post-processing to compute the power time series is faster.

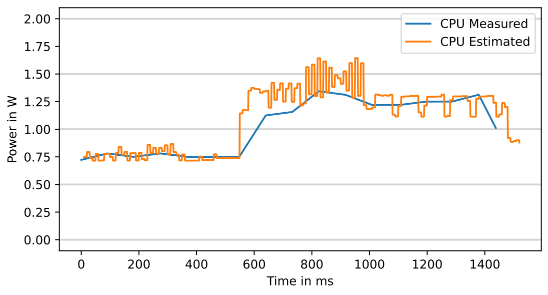
When using the pseudo-synchronization introduced in Section 4.2.1, we do not have power estimation samples until the accelerator is invoked. In a real system, the accelerator still draws static power though. We close this gap by briefly running a standalone xsim gate-level simulation of the accelerator, where we don't stimulate any of the accelerator's inputs besides the clock signal. This yields signal activities we can use to estimate idle power. While computing the power estimation time series for the FPGA, use this idle power draw during timespans where pseudo-synchronization is active.

4.3.2 *CPU and CPU Caches Power Estimation*

For the CPU power model, we use GemStone [46], which we introduce in Section 4.3.2. We initially planned to follow the GemStone methodology to build our own linear-regression-based power model from our physical ARM Cortex-A53. The authors used a different board with a 32-bit ARM Cortex-A15 compared to our 64-bit CPU. This meant we couldn't reuse their disk images, which would have already contained all workloads for training the linear



(a) resnet18-cpu



(b) resnet34-cpu

Figure 4.2: Power time series for CPU and caches measured vs. estimated using the adapted GemStone ARM Cortex-A15 power model. As a workload, we used TVM [3] running ResNet [15] image classification models on the CPU.

regression model ready to run. Due to limited time, we had to abort chasing down the sources Walker et al. used and getting them to cross-compile. Building an accurate power model for the CPU is also orthogonal to our work, in which we aim to show the feasibility of AC/DSim to combine existing simulators and power models to produce full-system energy and power estimates.

Instead, we decided to reuse the A15 power model that Walker et al. provided along with their paper on GemStone. Due to significant differences, we expect higher errors than GemStone's authors observed. Most prominently, the A15 is an out-of-order CPU, whereas our A53 is in-order. Further, the A15 uses the 32-bit ARMv7-A instruction set compared to the 64-bit ARMv8-A for the A53. In gem5, we also use an in-order model for the CPU instead of an out-of-order one like Walker et al. did. Together with updates to gem5, this means that the names of gem5 statistics changed across the board and possibly also their semantics.

To adapt the existing A15 power model, we updated the map from PMC events to gem5 statistics to incorporate the changed gem5 statistic names. We further manually adjusted the hard-coded voltage value and intercept until the power time series matched roughly what we got on our physical FPGA SoC board when running a resnet34-v1 image classification on the CPU only. More on these workloads in Section 5.2 in Chapter 5. We show the result in Figure 4.2. We acknowledge using the same class of workloads to adjust the CPU power model that we also use for our evaluation does inherently introduce overfitting issues. We stress though that building an accurate and stable power model that works even for unseen workloads is orthogonal to the goals of this thesis.

The GemStone authors also built a power model for a Cortex-A7 in-order CPU, which draws significantly less power than our A53 at peak though. We tried to adjust it in a similar manner but could not. When we pushed the Voltage too high, the dynamic power started shrinking.

To feed the adjusted GemStone power model with accurate workload information, we sample CPU and cache performance statistics in gem5, which gem5 already fully supports. gem5 provides the `m5` binary to give software running on the simulated host access to simulation control. Part of the functionality it provides is to reset the performance statistics or dump them, along with an option to dump periodically. So just before the workload of interest starts, we invoke `m5` to reset all statistics, create an immediate dump, which we use as a marker in the full-system power time series, so we know when the workload started, and instruct gem5 to dump statistics every sampling period

4.3.3 *Post-Processing for Full-System Energy Usage*

To produce a full-system power time series that we can use together with Equation 2.1 to compute full-system energy usage, we cannot just sum up the individual power models' time series because timestamps of the samples might not align. This happens because we, for example, only start dumping workload information in gem5 once the workload of interest starts to get a start marker and, similarly, stop dumping right after it finishes for a stop marker. Further, we allow the user to configure the sampling frequency per power model to trade off the amount of data collected and accuracy.

To solve this, we use constant interpolation (explained further down) to evaluate all individual power time series at a common-denominator sampling period regarding the sampling periods of all workload information involved. This allows us to simply sum the individual into a full-system estimate since this yields power estimates for the same timestamps across power models.

Concretely, we use 1 millisecond, allowing us to sum them into a full-system estimate, but this is adjustable by the user. All workload information samples for the power models we use are statistics, and represent averages over one sampling window. So given a power model that provides power samples 10, 20, 30 ms, we assign the same value for every integer millisecond in the range $[0, 9)$, $[10, 19)$, $[20, 29)$ using the sampled values at 10, 20, 30 ms, respectively. We call this *constant interpolation*.

We only compute the full-system power time series for timespans, where all power models provide samples. We call these timespans *evaluation window* and give an example using two imaginary power models in Figure 4.3. While the workload of interest is running, we assume that the user correctly set up all simulators and power models to provide power samples. Therefore, we can discard power samples outside the evaluation window. Note that the accelerator power model we use provides power estimates throughout the whole workload duration since we inject the idle power draw when pseudo-synchronization is active.

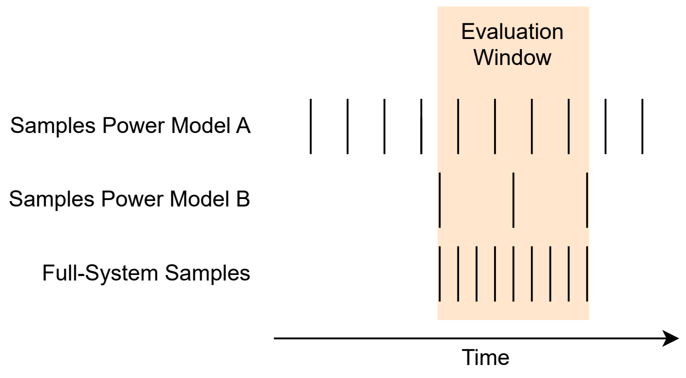


Figure 4.3: Power models A and B have different sampling frequencies and start/stop sampling at different timestamps. For computing the full-system power time series, we only consider the evaluation window, where both power models provide samples, and use a common denominator of all sampling periods as the evaluation sampling period.

5 EVALUATION

After outlining the implementation of AC/DSim for estimating power and energy of a simple FGPA SoC board, we now evaluate its accuracy in terms of the full-system energy estimate, as well as the overhead AC/DSim incurs both in terms of simulation speed and storage.

5.1 EVALUATION QUESTIONS

Concretely, we aim to answer the following evaluation questions:

RQ1 Can AC/DSim modularly combine existing power models and feed them with workload information to capture how power draw varies over time? (Section 5.5.1)

RQ2 How efficient is AC/DSim? Does AC/DSim incur simulation speed overhead due to the collection of workload information? How much data does AC/DSim need to collect, and how much time does it need for post-processing to produce the full-system energy estimates and power time series? (Section 5.5.2)

RQ3 (Can AC/DSim accurately estimate the energy usage of a system design and therefore enable meaningful evaluation of design choices when a physical testbed is unavailable? (Section 5.5.3))*

** For RQ3, we only provide preliminary results due to limited time.*

5.2 HW ACCELERATOR AND WORKLOADS

In this section, we present the open-source hardware accelerator we put on the FPGA and the workloads we use to evaluate the heterogeneous system in terms of full-system energy usage.

For the workload, use Apache TVM [3], an end-to-end machine learning compiler framework for CPUs, GPUs, and hardware accelerators. TVM can run image classification using popular deep neural networks like resnet50-v1 [15] either directly on the CPU or on a hardware accelerator published along with TVM named Apache VTA (Versatile Tensor Accelerator) [4]. Both TVM and VTA are fully open-source. Deep learning is a suitable workload for evaluating our framework since even when using the hardware accelerator, the CPU is still responsible for preparing the weights and inference data in memory. Some layers of the neural network's layers also still run on the CPU. This leads to non-trivial interactions in between CPU, host memory, and the accelerator. Full-system performance and energy usage are therefore not straightforward to predict with simple spreadsheet models.

The VTA accelerator is built to be deployed on an FPGA and offers configuration options to, for example, tune the size of its general matrix-multiply unit. This would allow us to deploy different hardware accelerator configurations and evaluate them in terms of full-system performance and energy usage. In practice, only the default configuration fits on the FPGA of our Avnet Ultra96-V2 (more info on this board in Section 4.1 and we decided to vary the clock speeds instead. 167.67 MHz is the fastest clock speed we can synthesize before seeing timing violations.

TVM itself is a mixture of Python code for the frontend and C++ code for the backend. The frontend is responsible for defining which model to run, which devices to use for the inference, and to set up the input data. Meanwhile, the backend handles performing the actual computations and compilation of models. Before we can perform inference for a pre-trained model with TVM, we need to invoke the TVM compiler to produce a shared library that contains the CPU and VTA instructions along with the model's weights. We compile all models beforehand so that the workload we measure only involves loading the model's shared library, loading and preparing the image to perform inference on, invoking the actual inference, and finally, fetching and printing the inference results. We show the output when running such an inference in Listing 5.1.

For evaluation, we use resnet18-v1 and resnet34-v1 as the inference models. The number indicates the number of layers used. The lower the number of layers, the faster the inference can be performed. We chose the two smallest resnetxx-v1 variants because gate-level simulation is generally slow. Simulating the accelerator for resnet18-v1 already takes days on our machine.

The less power-accurate but faster alternative is RTL-level simulation of the accelerator but when using Vivado's xsim for that, the accelerator was broken. The developers of VTA used Verilator [39] for testing and debugging, so the reason seems to be differences between how Verilator and xsim interpret and run RTL code. We also cannot use Verilator since it cannot write files optimized for storing signal activities (`.saif`). It can only write waveform files, which are significantly larger in size and which we need to convert because Vivado's power estimation only supports `saif`. Due to limited time, we decided not to pursue RTL-level simulation and evaluate using only gate-level simulation.

```
Rep 0: Requesting remote device 13_212_125 ns
Rep 0: Sending and loading model 111_746_748 ns
Rep 0: Pure inference duration 642_302_235 ns
Rep 0: End-to-end latency: 876_776_378 ns

prediction for sample 0
    #1:tiger cat 9.03182315826416
    #2:Egyptian cat 8.980079650878906
    #3:tabby, tabby cat 8.906598091125488
    #4:lynx, catamount 6.521409511566162
    #5:weasel 5.949103355407715
```

Listing 5.1: Terminal output of a TVM resnet18-v1 image classification inference when using only the CPU for computing. The input image was a cat. Along with the classification labels, TVM also outputs a value representing the certainty.

Sampling Frequency	Mean CPU Power [mW]	Std CPU Power [mW]
1 Hz	585.07	8.59
10 Hz	591.10	2.12
100 Hz	591.94	1.28
1000 Hz	592.43	2.11

Table 5.1: The power draw of our FPGA SoC board when idle and measuring power draw with `gemstone-profiler-logger` using different sampling frequencies. We show the mean power draw when measuring for 10 seconds, along with its standard deviation.

5.3 MEASURING THE PHYSICAL BASELINE

As the baseline to compare AC/DSim against, we use measurements from our physical system based on an Avnet Ultra96-V2, which we introduce in Chapter 4 in Section 4.1. We now discuss how we use the GemStone framework to automatically run workloads, collect the power measurements, and possible disturbing factors.

For measuring the FPGA SoC board, we rely on `gemstone-profiler-logger`, a C program provided by the GemStone framework that periodically reads the sensors by accessing their respective Linux character device files. For automating running and measuring workloads on our FPGA SoC board, we use another GemStone tool called `gemstone-profiler-automate`, which is a collection of Python scripts that use `gemstone-profiler-logger` to periodically collect the measurements with low overhead. These scripts also include post-processing to, for example, select the median run in terms of measured workload duration and write all collected information in a CSV file for easy further processing. The number of runs to perform is a command-line parameter when invoking one of the `gemstone-profiler-automate` scripts. We modified both `gemstone-profiler-logger` and `gemstone-profiler-automate` and publish our modifications on Github[1]

To be sure, we verify the low overhead claim of `gemstone-profiler-logger` by measuring our FPGA board while idle using different sampling frequencies. For the workload, we use `sleep 10`. We empirically determined that it does not make sense to collect sensor measurements at a higher frequency than roughly 12.5 Hz because the the Linux character device files are not updated more frequently than that. Still, we can read them at a higher rate. We show the results in Table 5.1. In general, all power draw means are within a standard deviation of the measurement for 1 Hz. The reason for 1 Hz showing a higher standard deviation is the limited resolution of our power sensor. Concretely, all power measurements either take the values 562.50 mW or 593.75 mW. The sensor cannot provide values in between. We conclude that measurements using `gemstone-profiler-automate` have indeed negligible overhead regarding power draw. For the evaluation of AC/DSim, we use our empirically determined maximal sampling frequency of 12.5 Hz.

Another influence on the power draw of our physical board is temperature. Under load, its temperature varies considerably as it only features a passive heatsink. GemStone can

1 Modifications to `gemstone-profiler-logger` and `gemstone-profiler-automate`: https://github.com/jonas-kaufmann/gemstone-profiler-logger/tree/acdsim and https://github.com/jonas-kaufmann/gemstone-profiler-automate/tree/acdsim

take the temperature into account when estimating power draw. Vivado is generally also capable of doing this. However, changing the junction temperature setting during power estimation doesn't influence the estimation for our concrete board. Therefore, we added a fan, which is enough to keep the temperature close to constant when comparing idle and load. To verify this, we sample the temperature alongside the power sensors.

Finally, we observe that the first run of a workload is always around 20 % slower than the following runs. This has a significant effect on the system's energy usage in the first run. We expect the reason for this to be the filesystem and CPU caches that need to warm up. To remove the influence of this and also try to minimize run-to-run influence, we always execute and measure each workload 7 times, throw away the first two runs, and then under the remaining, extract the median run regarding workload performance during post-processing. We use this median run for performance, power and energy for the results we report.

5.4 MEASUREMENTS WITH AC/DSIM

Similarly to measuring the physical baseline, we also take the effect of caches into account when running our workloads in the simulation. To warm up the caches, we invoke the whole workload application twice. For the first run, we don't log anything and also don't invoke the accelerator to simulate faster. Invoking the accelerator would not influence the caches since the accelerator accesses memory directly.

Further, SimBricks already handles minimizing run-to-run variance by replacing the init script using the `init` kernel command-line parameter. We only load what is necessary to run TVM and VTA. This approach is preferable over executing the workload multiple times since that is costly due to already slow simulation speeds.

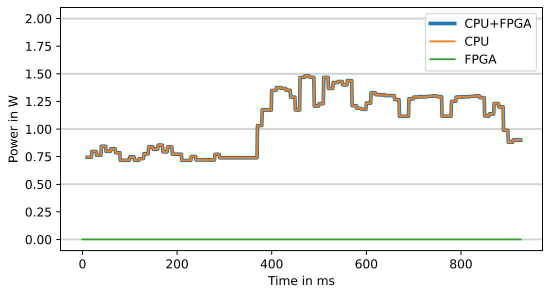
For the AC/DSim sampling frequency, we choose 10 ms across all power models, which is more accurate than the physical system's measurements, while keeping the workload information samples manageable (100 per virtual second).

5.5 RESULTS

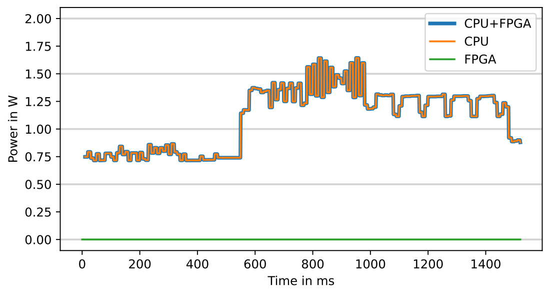
In this section, we present the experiments we ran to evaluate AC/DSim and discuss the results we obtained in the context of our evaluation questions (Section 5.1).

5.5.1 *RQ1: Feasibility of Modularly Combining Power Models*

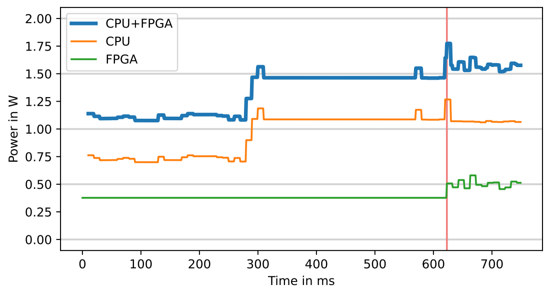
For building AC/DSim, two of our goals are to modularly combine existing power models and feed them with workload information such that we can produce full-system power estimates, be able to break that down into individual components while also capturing the influence on power draw of the software running in the system. To evaluate this, we ran 6 experiments in total, feeding the same image of a cat into resnet18-v1 and resnet34-v. We perform the inference using either only the CPU or CPU + VTA. We present the power time series that AC/DSim produces for the individual components and the full-system estimate (CPU+FPGA) in Figure 5.1.



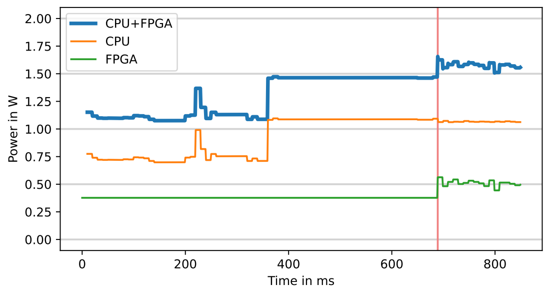
(a) resnet18-cpu



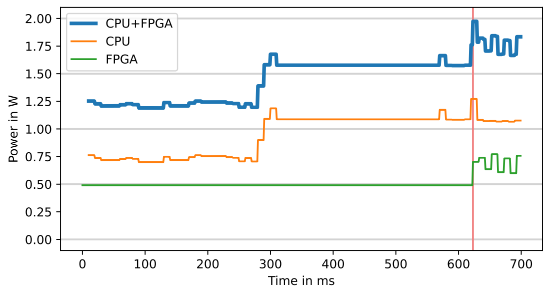
(b) resnet34-cpu



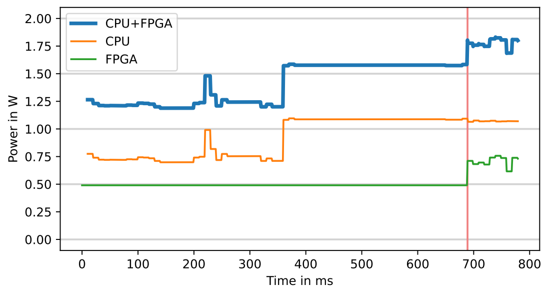
(c) resnet18-vta 100 MHz



(d) resnet34-vta 100 MHz*



(e) resnet18-vta 167 MHz



(f) resnet34-vta 167 MHz*

Figure 5.1: The power over time estimated by AC/DSim for CPU-only and VTA-accelerated inference using resnet18-v1 and resnet34-v1 models. The red vertical line marks the point in time when TVM invokes the accelerator. gem5 crashes before the workload completes for all workloads that involve VTA. (*) The resnet34-vta workloads did not finish after more than 9 days of simulation, so the power their power time series are cut off at the end.

Note that for all experiments that use VTA, gem5 crashes close to the end. The CPU-only workloads run all the way through. By comparing the simulation to our physical FPGA SoC board, we verified that the accelerator finishes all computations. gem5 seems to crash when TVM attempts to run one of the final layers on the CPU. From profiling the same workloads running on the physical SoC board, we know that the part of the workload that isn't executed as a consequence would only take an additional duration in the order of 10s of milliseconds though. We further discuss this when evaluating the accuracy of AC/DSim compared to the physical baseline in Section 5.5.3.

Looking at the CPU-only inference without digging into the internals of how TVM executes the models or comparing to measurements from the baseline physical system, the pattern of the CPU's power over time matches our expectations. First, the Python frontend script for TVM connects to the RPC server and loads the shared library that represents the compiled model. After invoking the model's shared library, TVM first loads the weights into memory. We call all this preparation phase. The preparation phase is single-threaded, whereas the CPU performs matrix multiplications on all cores during actual inference. During inference, we therefore expect a higher power draw than in the time before, which shows up in the power time series, although we do not know at which exact point in time the matrix multiplication starts to verify whether that indeed corresponds with the rise in power draw. Moving on to VTA-accelerated inference, CPU-wise, we see roughly the same power draw during the preparation phase, which likely rises after that due to the first layers being executed on CPU. From our optimization to avoid paying the simulation slowdown for gate-level simulation before the accelerator is invoked (Section 4.2.1), we know exactly when the workload starts using VTA. As expected, the power draw of the virtual FPGA rises the exact moment VTA is invoked.

Based on our observations, we answer RQ1 positively: AC/DSim can combine existing power models and feed them with workload information to capture power varying over time. Based on our understanding of the workload, the estimated power time series match the power trends we expect, indicating accurate workload information.

5.5.2 *RQ2: Estimation Efficiency*

For RQ2, we evaluate the storage and computation overhead to run AC/DSim compared to a SimBricks full-system simulation that doesn't involve any AC/DSim additions like collecting workload information. Further, we also quantify the post-processing overhead. We note that since AC/DSim uses SimBricks as the basis, we cannot simulate faster or consume less storage than a SimBricks simulation. Therefore, SimBricks is our comparison baseline.

We first report the storage overhead since we need that while discussing our results on simulation speed overhead. The storage size of the collected workload statistics depends on the sampling frequency per component. Concretely, gem5 periodically dumps all statistics into a text file. Meanwhile, xsim periodically writes an SAIF file. A single gem5 stats sample is 252 KB. For VTA, a single SAIF file, regardless of clock speed, is roughly 48.5 MB. For our experiments, we used a sampling frequency for both CPU and FPGA of 100 Hz. This means that, in total, simulating one virtual second requires 4.875 GB of storage space, which is very manageable. For xsim RTL-level simulations, the storage space required is even

Workload	SimBricks Slowdown	AC/DSim Slowdown	Overhead
resnet18-cpu	3'001	3'012	0.36 %
resnet34-cpu	3'550	3'550	0 %
resnet18-vta-100	7'070'294	10'291'565	45.56 %
resnet18-vta-167	10'917'214	16'740'019	53.34 %
resnet34-vta-100	5'933'606	11'340'959	91.13 %
resnet34-vta-167	9'712'335	17'659'776	81.83 %

Table 5.2: The simulation speed overhead when collecting the workload information we require for AC/DSim compared to running an identical SimBricks simulation without collecting any workload information. The slowdown factor for SimBricks and AC/DSim represents how many seconds we need to simulate to advance the virtual timestamp by one second. The overhead is computed as $\frac{\text{AC/DSim Slowdown}}{\text{SimBricks Slowdown}}$. The clock frequency of VTA is the last number in the workload name, i.e. 100 or 167, and is in MHz.

less since there are less signals to log due to the higher abstraction level. The storage overhead increases proportionally with the sampling frequency and also increases for more complex accelerators. Modeling additional components in gem5, for example, adding cores, also increases the space required, although the workload information for the hardware accelerator is far more significant in comparison.

We now discuss the simulation speed overhead results, which we present in Table 5.2. We ran all simulations for these results on a machine with 2x Intel Xeon Gold 6152 CPUs. Starting with the CPU-only workloads, we see that AC/DSim has negligible overhead over a SimBricks simulation. The same does not hold when also simulating the hardware accelerator. First of all, due to our optimization of not simulating the accelerator before it is invoked (Section 4.2.1), simulating up to the point of the accelerator's invocation is as fast as simulating the CPU-only workloads. In Table 5.2, we report the worst case when xsim fully simulates the hardware accelerator at gate-level. In that case, gate-level simulation is the bottleneck on the overall simulation progress with a higher than 1 million slowdown for the SimBricks baseline already.

In practice, this means that simulating a single virtual second takes more than 11 days, which rules out gate-level simulation for many workloads and also gave us a tough time during the evaluation. This clearly highlights that gate-level simulation is the bottleneck on overall simulation speed. Interestingly, the slowdown nearly doubles when collecting signal activities for AC/DSim. This overhead does not come from the size of the SAIF files, since these are in the order of 10s of Megabytes. Therefore, computing the signal activities has to be expensive. Looking at the SAIF file contents, xsim writes four statistics for every gate-level signal. We expect that these statistics have to be updated every clock cycle and for every signal.

In practice, even when simulating on a Ryzen 7 6800HS, which has a significantly higher single-threaded performance yielding roughly 3 times the simulation speed of the Intel server, we still had to simulate for multiple days even for the slowest workload resnet18, in which case the hardware accelerator was active for only 130 ms.

Finally, regarding post-processing, the only part that takes significant time is repeatedly invoking Vivado's power estimation algorithms. For resnet18-v1 and VTA running at 100 MHz, we have to process 15 SAIF files, which takes 452 seconds, which is roughly 30 seconds per SAIF file. So, similar to storage overhead, the post-processing time of AC/DSim is proportional to the sampling frequency of the hardware accelerator. Further, the more complex the hardware accelerator, the longer the power estimation in Vivado takes.

Based on our observations, we can answer RQ2: AC/DSim can efficiently combine existing power models and simulators to simulate provide full-system performance and energy / power estimates, although, depending on the concrete simulators used, there can be a significant simulation speed overhead for collecting detailed workload information.

5.5.3 *(Preliminary) RQ3: Accuracy of Power and Energy Estimates*

We now evaluate the accuracy of the component-level and full-system estimates that AC/DSim provides. For this, we compare with measurements taken on the physical FPGA SoC board (Section 4.1). We note that the results presented in this section are preliminary since gem5 crashes and our gem5 configuration also involves a modeling error of the FPGA SoC board, leading to significantly different durations for all hardware-accelerated workloads.

We start by comparing the workloads' duration observed in simulation against what we measure on our physical board. Errors here will also have an influence on the energy estimate produced by AC/DSim. The concrete results are in Table 5.3. For the workloads that only use the CPU, the error of the simulated duration is less than 10 %.

For workloads that involve VTA, the picture is more contrived. First of all gem5 crashes right after VTA finishes computing the last layer it is responsible for. We briefly dug into this and it seems that the CPU is sending cached accesses to the DMA region, whereas the accelerator uses uncached accesses. This triggers an assertion in the cache model. The fix is to map the DMA regions as uncached in the contiguous memory allocator we use. We did not have the time to re-run simulations though. Still, We can collect all workload information and measure the workloads' duration up until this crash. We measured the duration beginning at the invocation of VTA until the workload's end on our physical FPGA SoC and observed 143 ms for resnet18-v1 and 232 ms for resnet34-v1, irrespective of accelerator clock speed. So the part of the workload that follows after the crash cannot take longer than that. From the AC/DSim simulation, we know that the accelerator is active for roughly 130 ms. So, even accounting for gem5 modeling errors, the missing part is in the order of 10s of milliseconds.

Comparing the results for the individual workloads, we see that for AC/DSim, we severely undershoot the actual workload execution time. As we just argued, this cannot be explained by gem5 crashing while the workload is still running. We can rule out that this originates from VTA since it is TVM invokes it significantly later after on our physical platform compared to AC/DSim. The accelerator itself is invoked significantly later. In the physical system, this happens 1050 ms into the workload, which is independent of clock speed. Meanwhile, in simulation, TVM already invokes VTA after just 623 ms. This gap close to fully explains the difference in workload duration. Since before invoking VTA, the

Workload	Measured Median (Std)	Simulated Duration	Error
resnet18-cpu	0.846 (0.003) s	0.926 s	9.46 %
resnet18-vta 100 MHz	1.228 (0.003) s	0.749 s*	-39.01 %
resnet18-vta 167 MHz	1.229 (0.007) s	0.699 s*	-43.12 %
resnet34-cpu	1.439 (0.004) s	1.520 s	5.63 %
resnet34-vta 100 MHz	1.393 (0.012) s	DNF	DNF
resnet34-vta 167 MHz	1.4(0.006) s	DNF,	DNF

Table 5.3: Median workload duration and standard deviation measured on the physical FPGA SoC board in comparison to the simulated duration from AC/DSim. The last column shows the error relative to the physical baseline. For all workloads running in AC/DSim and involving VTA, gem5 crashed (*). In this case, we measured the duration of the workload until the crash. From profiling the physical system, we know that the part of the workload after the crash that is not executed is in the order of 10s of milliseconds.

The resnet34-vta workloads did not finish after more than 9 days of simulation.

workload runs fully on the CPU, we conclude that we are facing a performance modeling error in gem5. Due to time constraints, we did not look further into this.

Before invoking the accelerator, the workload duration is the same across VTA clock speeds when looking at the same model. Therefore, we can compare the numbers for `resnet 18-vta 100 MHz` and `resnet18-vta 167 MHz` and notice that the higher-clocking configuration is 50 ms faster. On the physical platform, VTA clock speed doesn't make a difference, which means that in practice, VTA's performance is limited by memory bandwidth. In the simulation, VTA's DMA accesses are handled in gem5. So here, we are again dealing with a gem5 modeling error regarding either the widths of the buses involved in handling a DMA access or the configuration of the memory controller.

Next, we look at the full-system energy estimates and how the power time series for the individual components compare. We present the results for estimated energy in Table 5.4. The gem5 modeling error causes high errors in the AC/DSim energy estimates for workloads involving VTA. In comparison, the CPU energy estimates have an error below 15 %.

Assuming we wanted to build a system optimized for maximum energy, the estimates from AC/DSim suggest that the CPU-only system is the best, although only ever so slightly. When comparing this with the measurements on the physical platform, we draw the same conclusion, for resnet18-v1, although energy usage, in reality, is actually much better for the CPU-only configuration. Since the energy distance between CPU-only and VTA-accelerated configurations decreases for larger model sizes, we expect that the VTA-accelerated configuration would win for resnet50-v1.

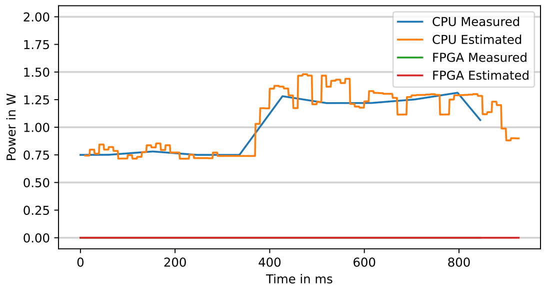
We can further break down the full-system energy estimate errors by looking at the components' individual power draw over time, which we present in Figure 5.2. We see that the estimation for the CPU power time series tends to generally overshoot. This is due to adopting a power model built for a different CPU using simple adjustments (Section 4.3.2). For the FPGA, however, we can see that despite the best case, since we use gate-level simulation together with the manufacturer's power estimation models, generally overshoots.

Workload	Measured Energy (Std)	Simulated Energy	Error
resnet18-cpu	0.848 (0.019) J	0.972 J	14.58 %
resnet18-vta 100 MHz	1.389 (0.01) J	1.001 J*	-27.95 %
resnet18-vta 167 MHz	1.369 (0.029) J	1.006 J*	-26.48 %
resnet34-cpu	1.492 (0.024) J	1.662 J	11.41 %
resnet34-vta 100 MHz	1.611 (0.024) J	DNF	DNF
resnet34-vta 167 MHz	1.6 (0.051) J	DNF	DNF

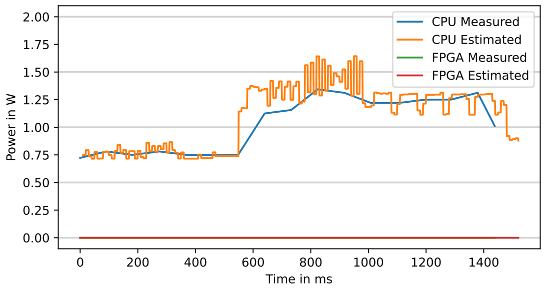
Table 5.4: Full-system energy usage and standard deviation measured on the physical FPGA SoC board in comparison to the AC/DSim estimation. The last column shows the error relative to the physical baseline. For all workloads running in AC/DSim that involve VTA, gem5 crashes and we additionally identified significant modeling errors (*). The resnet34-vta workloads did not finish after more than 9 days of simulation.

Applying a simple constant offset would bring the physical baseline and estimation closer together though. The limited resolution of the power sensor makes Comparing VTA at 100 MHz to 167 MHz, we see that the FPGA power model reports a higher peak power draw for the higher clocking configuration, which we can also observe in the physical testbed. As discussed earlier in this section, the computation on VTA in the physical system is limited by memory bandwidth, whereas in simulation it is not and can therefore finish faster. This means that VTA's signal activities are higher in simulation than in the physical testbed, which also leads to higher estimated power draw under load. When correcting this and applying a constant offset to the FPGA power model, we believe we could make the FPGA power estimates very accurate.

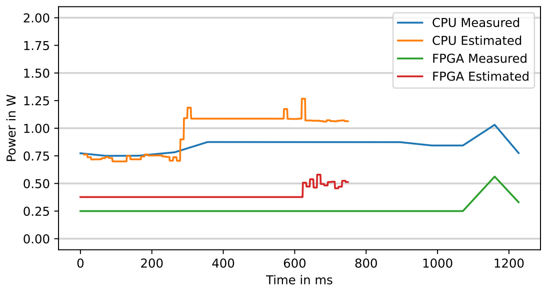
Based on our observations, we can tentatively answer RQ3: Under the assumption that the user carefully validates the simulators and power models used in AC/DSim against the physical system where possible, we believe that AC/DSim can be used to reliably make good or even optimal design choices. When skipping validation, however, AC/DSim can be very inaccurate, leading users to the wrong design choices.



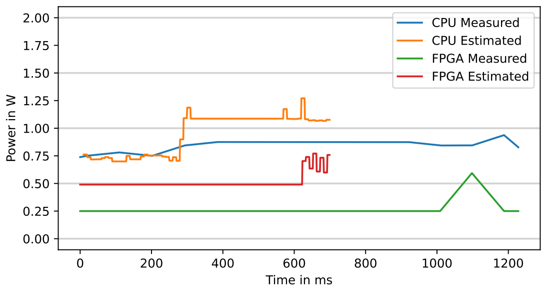
(a) resnet18-cpu



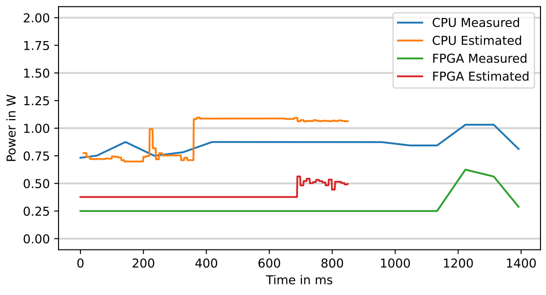
(b) resnet34-cpu



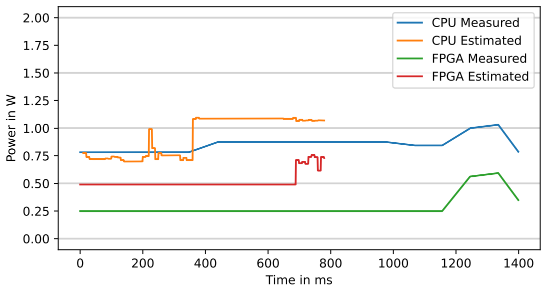
(c) resnet18-vta 100 MHz



(d) resnet18-vta 167 MHz



(e) resnet34-vta 100 MHz*



(f) resnet34-vta 167 MHz*

Figure 5.2: Comparison of component-level power draw over time as estimated by AC/DSim with the measurements taken on our physical FPGA SoC. (*) The resnet34-vta workloads did not finish after more than 9 days of simulation so the power time series are cut off at the end.

6 RELATED WORK

After evaluating AC/DSim and discussing the results, limitations, and future steps, we now present related work.

AC/DSim is based on the full-system simulation tool SimBricks [24], which combines and connects models for individual components to capture interactions between them. An alternative to SimBricks is SST(Structural Simulation Toolkit) [35], which also aims to connect existing models using standardized interface. SimBricks already implemented support for the simulator combinations we required though, supporting gem5 [6, 26] for the host, a selection of NIC models, and network simulators such as OMNeT++ [45] and ns-3 [32]. Especially the integration of these accurate network simulators enables AC/DSim to grow to larger systems involving multiple hosts without additional modifications as long as the user is not interested in power or energy models for the network.

Compared to these very flexible full-system simulation tools, Simics [27] also claims to be full-system. This may be true since it allows you to instantiate multiple hosts, connect them over a network, and introduce additional models by using their included API. However, the closed-source nature of Simics severely limits its flexibility. Simics also does not feature a proper cache model. The existing cache model can only inject latencies but does not return the actual data. We therefore cannot extract the workload information we require for our CPU power estimation models based on GemStone [46], which requires performance statistics for the caches. We expect issues for using other power models as well.

Aside from full-system simulation tools, ad-hoc solutions for specific types of systems that include energy estimation exist. For accelerator-rich processor architectures, for example, PARADE [11] provides full-system performance and power estimates for general-purpose and specialized cores. It does so by combining RTL-level simulation and power estimation for the specialized cores, both performed using Synopsys Desgin Compiler [41], and CPU simulation for the general-purpose cores using gem5. For the CPU power estimation, the authors rely on McPAT [25]. Except for the CPU power model, this is similar to the approach we used for AC/DSim. Although PARADE can estimate power for the full system, it is limited to small systems of the class accelerator-rich processor architecture.

MofySim [29] is another tool for evaluating performance and energy usage of a specific class of system, in this case mobile phones. Again, it combines gem5 with McPAT for the CPU and simple linear models for network and display.

McPAT [25] in alternative to the linear-regression based power model we use in this thesis. It takes a hierarchical approach for modeling the power consumption of processors. The user has to provide detailed configuration parameters for the CPU that they want to model, parameterizing cores, NoCs, caches, memory controllers, etc. Similar to our implementation, McPat also takes workload information as input, for example, from a gem5 simulation. To estimate power, it hierarchically decomposes the high-level blocks into basic circuit blocks. It then uses analytic models for each block of these low-level blocks to estimate power, taking additional configuration parameters such as the manufacturing technology node into account.

Compared to AC/DSim, however, using McPAT requires a deep understanding of the CPU and its internal layout. Aside from possibly not having access to such detailed information, we demonstrated in our evaluation that this can easily lead to significant modeling errors. In comparison, AC/DSim allows the user to use linear regression power models such as GemStone [46] instead, which do not require any configuration parameters and are by design hardware-validated since they are built using the physical CPU the system design targets.

Aside from McPAT, Aladdin [37] is an interesting candidate for replacing cycle-accurate RTL or gate-level simulation and the Vivado power estimation we used. Aladdin is a trace-based accelerator simulator that profiles the dynamic execution of a C program and constructs a dynamic data dependence graph as a dataflow representation of an accelerator. It then applies optimizations as well as constraints to the graph to create a realistic model of accelerator activity. The authors validated its accuracy to be within 7% accuracy compared to standalone RTL designs of the accelerator and 100 times faster than actual RTL simulation. gem5-Aladdin [38] already combines aladding aladdin with gem5 to capture the interactions between accelerator and CPU, although it uses gem5's syscall emulation instead of the full system simulation mode. Compared to McPAT, which just estimates power, Aladdin takes both roles, acting as the simulator and the power model. We have therefore have to further verify whether we can actually integrate Aladdin into our framework. The issue is the SimBrics full-system simulation, in which the modeled accelerator has to behave one one hand accurately for accurate full-system performance results and on the other, functionally correct.

Aside from related work presented in this section, we also highlight the integration of power models from prior work into AC/DSim when discussing future work in Section 7.2 to enable power draw estimation for additional components, such as memory, SSDs, and the network.

7 CONCLUSION

After evaluating AC/DSim, we now discuss our results in the context of the design goals we set in Section 3.1 and the limitations our implementation and evaluation face. We then highlight steps for future work.

7.1 DISCUSSION AND LIMITATIONS

When building AC/DSim, we set four design goals: full-system, modular, accurate, and scalable. We now discuss the results and the limitations we identified in the context of these.

During our evaluation, we showed that we can indeed modularly combine existing power models and simulators to produce full-system energy and power estimates. Although our implementation covered three components (CPU, caches, and FPGA) and therefore demonstrated the feasibility of estimating energy and power for the full system, we also need to consider system memory when evaluating design choices. For our physical test system (Avnet Ultra96-V2), for example, we measured up to 343.75 mW power draw just from the system memory, which we currently do not capture.

Regarding accuracy, we used a very limited selection of just inference workloads, all from the same family of deep-learning models. Therefore, our evaluation is not extensive enough to claim that AC/DSim can be reliably used to explore design choices in heterogeneous systems. We demonstrated the feasibility of efficiently combining existing power models and simulators for individual components though, which represents the first steps in this direction.

Regarding our workload choice, we could only evaluate small workloads that use the hardware accelerator for a maximum of 250 ms due to very slow gate-level simulation. Together with the limited resolution and sampling frequency of the power sensors of our physical FPGA SoC system, this introduces measurement errors for our physical baseline. We are able to sample the power sensors at an interval of 80 ms, which, even in the best case, means three samples while the hardware accelerator is under load. Further, this coarse temporal resolution means that the power measurements in the physical system cannot capture dynamic changes in accelerator load and, consequently, power draw that happens at a finer granularity than the power sensors' sampling frequency. On the other hand, this highlights a big advantage for AC/DSim and simulation in general: The user can freely decide the amount of detail they want to log. In the case of AC/DSim, the user can, for example, control the sampling frequency per component, enabling very fine-grained energy usage optimization.

We want to stress that, the user needs to validate the simulators and power models used against a physical system for the components for which this is possible. Otherwise, the estimates produced by AC/DSim can lead to bad system design choices, as we found during our evaluation in Section 5.5.3.

Finally, we discuss scalability. Our approach can be used to evaluate larger systems that involve, for example, multiple hosts with multiple hardware accelerators and a network in between. This is handled by SimBricks [24], the full-system simulation framework we use. Its authors demonstrated that given enough CPU cores, these larger simulations run with minimal impact on simulation speed compared to simulating smaller systems. The second part of AC/DSim is applying the power models to the sampled workload information. Since we do this in post-processing, this part is also straightforward.

However, slow simulators hinder scalability. In our case, gate-level simulation of the FPGA suffers from a more than 1'000'000 slow down, meaning we had to simulate for days just to advance virtual time by 100 ms. This is infeasible for large systems, which likely need to be simulated for tens of virtual seconds. We pick this up in the next section, presenting alternatives when discussing future work.

7.2 FUTURE WORK

After discussing our approach and highlighting the limitations during implementation, we now outline necessary future work to enable AC/DSim to reliably estimate the power draw and energy usage even of large systems.

The most obvious is making AC/DSim *more full-system* by adding additional power models and, if necessary, also implementing SimBricks support for additional simulators that can provide the necessary workload information to these power models. Interesting components to look into in the future are system memory, for example, using the DRAM controller model built into gem5 [14] and combining it with DRAMPower [8]. Another component that can use a significant amount of energy are SSDs [12], which FlashPower [30] can estimate. Larger Systems with multiple hosts involve networking, which can also use noticeable amounts of energy [30]. SimBricks already supports network simulators such as OMNeT++ [45] and ns-3 [32]. In the future, we could combine these with simple linear power models for network switches, routers, and wireless links such as those presented in [13, 20, 29, 34].

As highlighted in Section 7.1, we also need faster simulators for the hardware accelerator. Concretely, we believe that the slowdown should be at most 10'000, which means that 40 virtual seconds take roughly 5 days to simulate. In the case of VTA, this slowdown is achievable by RTL-level simulators. One such simulator is Verilator [39], which we found is able to simulate VTA running at 100 MHz with a slowdown of less than 1'000. For Vivado's RTL-level simulation, we could not measure the slowdown due to VTA's broken behavior in that simulator. Even though we expect Verilator to be faster than Vivado RTL-level, it produces huge waveform files. We can optimize this by trading off accuracy and only logging for a limited window every sampling period. When producing the hardware accelerator's power time series, we then assume that signal activities remain constant throughout one sampling period.

Finally, a direction we have not explored yet for AC/DSim is modeling temperature. Temperature and power draw interact with temperature causing higher power draw [31] and vice-versa, although this will reach an equilibrium in practice due to cooling or passive heat dissipation. We can almost double our FPGA's idle power draw by heating it up from 38 to 70 °C. Temperature estimation should therefore be integrated into the power models.

In fact, the GemStone framework [46], which we use for our CPU power model, already supports taking temperature as an input, although it cannot output temperature so this has to be handled by a separate model, for example, the power and temperature model built into gem5 [40]. Similarly, Vivado's power model generally takes the configured heatsink size into account and outputs a junction temperature, which has an influence on power estimation [2] However, this functionality seems to be disabled in our case, as no matter what we configure for the heatsink, the junction temperature and power draw remain the same.

BIBLIOGRAPHY

[1] Advanced Micro Devices, Inc. *AMD Vivado™ Design Suite*. https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vivado.html. Retrieved Nov 24, 2024.

[2] Advanced Micro Devices, Inc. *Vivado Design Suite User Guide: Power Analysis and Optimization (UG907 v2024.2)*. https://docs.amd.com/r/en-US/ug907-vivado-power-analysis-optimization.

[3] Apache Software Foundation. *Apache TVM - An End to End Machine Learning Compiler Framework for CPUs, GPUs and accelerators*. https://tvm.apache.org/. Retrieved Nov 27, 2024.

[4] Apache Software Foundation. *Apache VTA (Versatile Tensor Accelerator) - Open, Modular, Deep Learning Accelerator*. https://github.com/apache/tvm-vta. Retrieved Nov 27, 2024.

[5] Arm Limited. *AMBA AXI Protocol Specification*. https://developer.arm.com/documentation/ihi0022/latest/. Version September 2023.

[6] Nathan L. Binkert et al. "The gem5 simulator." In: *SIGARCH Comput. Archit. News* 39.2 (2011), pp. 1–7.

[7] Cadence. *Genus Synthesis Solution*. https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/synthesis/genus-synthesis-solution.html. Retrieved Nov 17, 2024.

[8] Karthik Chandrasekar, Christian Weis, Yonghui Li, Sven Goossens, Matthias Jung, Omar Naji, Benny Akesson, Norbert Wehn, and Kees Goossens. *DRAMPower: Open-source DRAM Power & Energy Estimation Tool*. http://www.drampower.info.

[9] David Cock et al. "Enzian: an open, general, CPU/FPGA platform for systems software research." In: ASPLOS '22: 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 434–451.

[10] Maxime Colmant, Romain Rouvoy, Mascha Kurpicz, Anita Sobe, Pascal Felber, and Lionel Seinturier. "The next 700 CPU power models." In: *Journal of Systems and Software* 144 (2018), pp. 382–396.

[11] Jason Cong, Zhenman Fang, Michael Gill, and Glenn Reinman. "PARADE: A cycle-accurate full-system simulation Platform for Accelerator-Rich Architectural Design and Exploration." In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD). 2015, pp. 380–387.

[12] Miyuru Dayarathna, Yonggang Wen, and Rui Fan. "Data Center Energy Consumption Modeling: A Survey." In: *IEEE Communications Surveys & Tutorials* 18.1 (2016), pp. 732–794.

[13] Martin Dräxler, Frederic Beister, Stephan Kruska, Jörg Aelken, and Holger Karl. "Using OMNeT++ for Energy Optimization Simulations in Mobile Core Networks." In: Fifth International Conference on Simulation Tools and Techniques. 2012.

[14] Andreas Hansson, Neha Agarwal, Aasheesh Kolli, Thomas Wenisch, and Aniruddha N. Udipi. "Simulating DRAM controllers for future system architecture exploration." In: 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 201–210.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. Dec. 10, 2015. URL: http://arxiv.org/abs/1512.03385.

[16] https://github.com/ultraembedded. *ultraembedded - AXI-4 JPEG Decoder*. https://github.com/ultraembedded/core_jpeg_decoder. Retrieved Nov 21, 2024.

[17] Intel Corporation. *Quartus® Prime Pro Edition User Guide: Power Analysis and Optimization (UG-20141)*. https://www.intel.com/content/www/us/en/docs/programmable/683174.html. Retrieved Nov 21, 2024.

[18] Jiaxin Lin, Kiran Patel, Brent E. Stephens, Anirudh Sivaraman, and Aditya Akella. "PANIC: A {High-Performance} Programmable NIC for Multi-tenant Networks." In: 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20). 2020, pp. 243–259.

[19] Norman P. Jouppi et al. "In-Datacenter Performance Analysis of a Tensor Processing Unit." In: ISCA '17: The 44th Annual International Symposium on Computer Architecture. 2017, pp. 1–12.

[20] Fabian Kaup, Sergej Melnikowitsch, and David Hausheer. "Measuring and modeling the power consumption of OpenFlow switches." In: 10th International Conference on Network and Service Management (CNSM). 2014, pp. 181–186.

[21] Dario Korolija, Timothy Roscoe, and Gustavo Alonso. "Do OS abstractions make sense on FPGAs?" In: 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20). USENIX Association, 2020, pp. 991–1010.

[22] Ian Kuon and Jonathan Rose. "Measuring the gap between FPGAs and ASICs." In: FPGA06: ACM/SIGDA International Symposium on Field Programmable Gate Arrays. 2006, pp. 21–30.

[23] Bojie Li, Zhenyuan Ruan, Wencong Xiao, Yuanwei Lu, Yongqiang Xiong, Andrew Putnam, Enhong Chen, and Lintao Zhang. "KV-Direct: High-Performance In-Memory Key-Value Store with Programmable NIC." In: SOSP '17: ACM SIGOPS 26th Symposium on Operating Systems Principles. 2017, pp. 137–152.

[24] Hejing Li, Jialin Li, and Antoine Kaufmann. "SimBricks: end-to-end network system evaluation with modular simulation." In: SIGCOMM '22: ACM SIGCOMM 2022 Conference. 2022, pp. 380–396.

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

[26] Jason Lowe-Power et al. "The gem5 Simulator: Version 20.0+." In: *CoRR* abs/2007.03152 (2020).

[27] Peter S Magnusson, Magnus Christensson, Jesper Eskilson, Daniel Forsgren, Gustav Hallberg, Johan Hogberg, Fredrik Larsson, Andreas Moestedt, and Bengt Werner. "Simics: A full system simulation platform." In: *Computer* 35.2 (2002), pp. 50–58.

[28] Max Planck Institute for Software Systems and National University of Singapore. *SimBricks Developer Guide - Architectural Overview*. `https://simbricks.readthedocs.io/en/latest/devel/arch.html`. Retrieved Nov 29, 2024.

[29] Minho Ju, Hyeonggyu Kim, and Soontae Kim. "MofySim: A mobile full-system simulation framework for energy consumption and performance analysis." In: IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). 2016, pp. 245–254.

[30] V. Mohan, T. Bunker, L. Grupp, S. Gurumurthi, M. R. Stan, and S. Swanson. "Modeling Power Consumption of NAND Flash Memories Using FlashPower." In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32.7 (2013), pp. 1031–1044.

[31] Yehya Nasser, Jordane Lorandel, Jean-Christophe Prevotet, and Maryline Helard. "RTL to Transistor Level Power Modeling and Estimation Techniques for FPGA and ASIC: A Survey." In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40.3 (2021), pp. 479–493.

[32] nsnam. *ns-3 | a discrete-event network simulator for internet systems*. `https://www.nsnam.org/`. Retrieved Feb 2, 2022.

[33] Saeed Rashidi, Srinivas Sridharan, Sudarshan Srinivasan, and Tushar Krishna. "ASTRA-SIM: Enabling SW/HW Co-Design Exploration for Distributed DL Training Platforms." In: *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2020, pp. 81–92.

[34] P. Reviriego, V. Sivaraman, Z. Zhao, J. A. Maestro, A. Vishwanath, A. Sanchez-Macian, and C. Russell. "An energy consumption model for Energy Efficient Ethernet switches." In: International Conference on High Performance Computing & Simulation (HPCS). 2012, pp. 98–104.

[35] A. F. Rodrigues et al. "The structural simulation toolkit." In: *SIGMETRICS Perform. Eval. Rev.* 38.4 (2011), pp. 37–42.

[36] Hugo Sadok, Aurojit Panda, and Justine Sherry. "Of Apples and Oranges: Fair Comparisons in Heterogenous Systems Evaluation." In: HotNets '23: The 22nd ACM Workshop on Hot Topics in Networks. 2023, pp. 1–8.

[37] Yakun Sophia Shao, Brandon Reagen, Gu-Yeon Wei, and David Brooks. "Aladdin: a Pre-RTL, power-performance accelerator simulator enabling large design space exploration of customized architectures." In: *ACM SIGARCH Computer Architecture News* 42.3 (2014), pp. 97–108.

[38] Yakun Sophia Shao, Sam Likun Xi, Vijayalakshmi Srinivasan, Gu-Yeon Wei, and David Brooks. “Co-designing accelerators and SoC interfaces using gem5-Aladdin.” In: 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). 2016, pp. 1–12.

[39] Wilson Snyder, Paul Wasson, and Duane Galbi. *Verilator*. `https://verilator.org`. Retrieved Nov 27, 2024.

[40] Vasileios Spiliopoulos, Akash Bagdia, Andreas Hansson, Peter Aldworth, and Stefanos Kaxiras. “Introducing DVFS-Management in a Full-System Simulator.” In: IEEE 21st International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS). 2013, pp. 535–545.

[41] Synopsys, Inc. *Design Compiler*. `https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html`. Retrieved Dec 16, 2024.

[42] Synopsys, Inc. *PrimePower - RTL to Signoff Power Analysis*. `https://www.synopsys.com/implementation-and-signoff/signoff/primepower.html`. Retrieved Nov 22, 2024.

[43] Tao Wang, Xiangrui Yang, Gianni Antichi, Anirudh Sivaraman, and Aurojit Panda. “Isolation Mechanisms for High-Speed Packet-Processing Pipelines.” In: 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22). 2022, pp. 1289–1305.

[44] The OpenROAD Project. *OpenSTA*. `https://github.com/The-OpenROAD-Project/OpenSTA/blob/master/doc/OpenSTA.pdf`. Retrieved Nov 22, 2024.

[45] András Varga and Rudolf Hornig. “An overview of the OMNeT++ simulation environment.” In: *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*. 2008.

[46] Matthew Walker, Sascha Bischoff, Stephan Diestelhorst, Geoff Merrett, and Bashir Al-Hashimi. “Hardware-Validated CPU Performance and Energy Modelling.” In: IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). 2018, pp. 44–53.

[47] Matthew J. Walker, Stephan Diestelhorst, Andreas Hansson, Anup K. Das, Sheng Yang, Bashir M. Al-Hashimi, and Geoff V. Merrett. “Accurate and Stable Run-Time Power Modeling for Mobile and Embedded CPUs.” In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36.1 (2017), pp. 106–119.

[48] Zhizhen Zhong, Mingran Yang, Jay Lang, Christian Williams, Liam Kronman, Alexander Sludds, Homa Esfahanizadeh, Dirk Englund, and Manya Ghobadi. “Lightning: A Reconfigurable Photonic-Electronic SmartNIC for Fast and Energy-Efficient Inference.” In: ACM SIGCOMM ’23: ACM SIGCOMM 2023 Conference. 2023, pp. 452–472.