Journal of Systems Architecture 59 (2013) 603–614

Contents lists available at SciVerse ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

An automatic energy consumption characterization of processors using ArchC

Marcelo Guedes *, Rafael Auler, Liana Duenha, Edson Borin, Rodolfo Azevedo

Computer System Laboratory – LSC, University of Campinas – UNICAMP Campinas, São Paulo 13083-852, Brazil

ARTICLE INFO

Article history:
Available online 13 June 2013

Keywords:
ArchC
acSynth
acSim
acPower
acPowerGen
Processor
High-level
Characterization
Power
Consumption
Simulation

ABSTRACT

The design complexity of integrated circuits requires techniques that automate and ease common tasks, allowing developers to keep up with the rapid growth and demand of the industry. This paper presents acSynth, an integrated framework for development and synthesis based on ArchC ADL descriptions and introduces a new power characterization method at the architectural level. The acSynth is composed of a characterization tool used to extract the energy consumption behavior of processors and also of a simulation method that, after characterization, is able to estimate software energy consumption at high speeds. Our experimental results show the power figures obtained by the characterization flow of Plasma and Leon3 processors, a MIPS-I and SPARCv8 HDL descriptions, respectively, using two different synthesis tools: Xilinx Xpower and Altera PowerPlay. The acSynth simulator used these power figures to allow power analysis at more than 35 million instructions per second in a simulation with small accuracy diversion and without loss of generality. The system executes large tests in minutes, which would otherwise take years in standard HDL methodologies.

1. Introduction

The design of a System-on-a-Chip (SoC) requires thorough assessing of the modules that compose it. In fact, designers must evaluate the combination of these components in order to select one from the best subset and discard those from the worst. Furthermore, to reduce system development costs and time, designers are increasingly using extra processor cores as components, thus transferring the implementation of hardware features to software. This increase in core usage for different software workloads requires careful selection of the cores that will be available to run the software implementation. Besides, designers have a myriad of processor variations to choose from. Due to the lack of tools to help the choice of the best design, the designer depends solely on her experience rather than solid design space exploration, leading to potentially inefficient overall system designs.

When designing a system, it is critical to know the energy consumption of an extra core. Not only the best choice requires detailed analysis of consumption profiles of different processors, but a given processor, depending on its power requirements, also impacts the choice of the other components in the system. Hence it is important to perform the design space exploration of processor alternatives. On the other hand, calculating the energy consumption of processors when running a given benchmark requires long simulation times. The crux of the problem is that traditional methods requires the simulation of the hardware at the gate level, after complete design workflow, since the energy consumption is dependent upon the final chip layout. Because it is inherently slow to tally the energy used in every signal transition, one efficient strategy to increase power estimation speed is to back-annotate energy information on a fast simulator devoid of circuit detail.

In this paper we define a precise methodology for extracting the energy consumption characteristics of processors on a per-instruction basis, which allows us to augment a fast Instruction Set Simulator (ISS) with energy information to efficiently provide power reports. We also leverage a set of Architecture Description Language (ADL) tools to perform this task automatically in an architecture agnostic approach, creating the `acSynth` workflow. `AcSynth` aids the designer in the task of choosing the best core for the system needs, enabling processor energy consumption characterization through ArchC [1–4], a SystemC-based ADL that easily integrates with other hardware components to compose the full system.

Designing with ADLs is easier because it describes a processor at a high level, allowing our method to automatically generate effective benchmarks for power estimation tailored to stimulate a specific architecture from the software perspective. After testing

* Corresponding author. Tel.: +55 (19) 3521-5857.
E-mail addresses: marcelo.guedes@lsc.ic.unicamp.br (M. Guedes), rafael.auler@lsc.ic.unicamp.br (R. Auler), liana.duenha@lsc.is.unicamp.br (L. Duenha), edson@ic.unicamp.br (E. Borin), rodolfo@ic.unicamp.br (R. Azevedo).

1383-7621/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.sysarc.2013.05.025

the benchmarks, we chose to augment the ArchC instruction set simulator with power information. Although other instruction-level power estimation approaches that use ADLs have been proposed in literature [5,6], they are semi-automatic. This paper extends Ma et al. [6] and Guedes et al. [7] work to provide an unsupervised, automatic, workflow for easy system design space exploration across different processor architectures, including multicore architectures. We test our method in 3 different processor-technology scenarios to show its feasibility.

Our key contributions are:

1. We present a new power characterization method at the architectural level.
2. We show how to automatically generate the characterization programs using ADL information, leading to an effective benchmark for processor power estimation.
3. We show a proof of concept of this methodology, applying our framework to the Plasma MIPS-I and SPARCv8 Leon3 processor, demonstrating its effectiveness.
4. We integrated the existing ArchC simulator generator with PowerSC library, enabling ArchC power-aware simulation.
5. We developed and reported results for a framework that is adaptable in the characterization and report generation process, allowing it to assess a wide range of processors.

This paper is organized as follows. Section 2 discusses the related work; Section 3 presents an end-to-end acSynth framework overview, beginning at the low-level power characterization data until the power model exploration; Section 4 provides experimental results for the framework and discusses power reports for the MIPS Plasma and SPARCv8 Leon3 processors using the Mediabench and Mibench benchmarks with large inputs and acStone benchmarks comparison with a reference power analysis software; finally, Section 5 presents our conclusions.

2. Related work

The power wall is one of the main problems that a system designer faces and we aim to help him by automatically creating a power model for the processor. Power estimation may be performed at several levels, each one involving different trade-offs of accuracy and speed: transistor level [8], gate level [9], cycle accurate level [10], instruction set level [5], which includes this work, and application level [11]. Previous methodology try to create a much more detailed power model by using all the hardware components [12,13], other techniques restrict the power figures to only a few processor states [14], while others create statistical power models for a specific workload [15]. We stay in the middle by using an instruction based power model for processor cores based on the work of Tiwari et al. [16], making it suitable for evaluating large software without a significant impact on precision.

Degalahal et al. [17] point the increasing FPGA market relevance due to the low non-recurring engineering costs, fitting FPGA solutions ideal for low-to-mid volume products. However, in comparison to an ASIC, FPGAs uses more transistor for equivalent hardware modules. This factor increases the importance of an accurate power analysis of FPGAs. They developed a tool for this specific case. They used device-level simulation for architectural modules characterization, avoiding silicon characterization, and then, simulated benchmark modules and gathered typical FPGA resources utilization. Each resource block was characterized as a single effective capacitance and, by crossing the characterized capacitance and the typical average switching activity of 12.5%, they found the subject FPGA typical design consumption.

Gupta et al. [5] present the closest work to our own and developed a methodology called Power-ArchC that also uses the ArchC ISS augmented with per instruction power values obtained by characterization. Nevertheless, their workflow is not applicable to a wide range of processors because it is not possible to adapt their characterization software automatically to different ISAs, a task that would require an automatic compiler generator since their characterization software is a benchmark written in a high level language. Our automatic approach, on the other hand, does not depend on the availability of a cross-compiler that targets the processor under test and thus could be even used as part of the design flow of a new processor ISA that lacks such software support. In contrast to our automatic approach, three instruction level power characterization (ILPC) campaigns were selected based on software benchmarks. They compared the energy at instruction level based on characterized data with energy at gate level simulation and the maximum error introduced with the ISS technique was 21%.

3. The acSynth framework

Our power estimation methodology through the acSynth framework integrates the following ArchC tools:

acSim is a simulator generator for processors described using the architecture description language (ADL) ArchC. An ArchC processor model comprises an instruction set architecture description file (ISA), and an architectural description file (ARCH) informing state storage elements and cache configuration. The tool uses this high-level description to generate a SystemC model. SystemC is a system description language (SDL) whose models can be compiled with the help of a C++ compiler to generate efficient hardware simulators. Rigo et al. [4] provide an extensive description of the ArchC ADL model and its simulator.

acPower is a power estimation tool developed by Ma et al. [6] that generates power reports from software running over the ArchC simulator. The acPower tool is a Tiwari's Method application embedded into the ArchC environment. This work was extended to interface with acSynth [18] and PowerSC [19].

acPowerGen is our automatic characterization code generator that provides programs required to extract power information from RTL.

ArchC and acSim are open-source and publicly available. The acPowerGen, acPower and acSynth will be available through the ArchC website [1].

3.1. Overview

ArchC has been designed at the Computer Systems Laboratory (LSC) in the Institute of Computing at the University of Campinas. The acSynth tool integrates several ArchC tools, providing a framework capable of performing fast high-level power analysis.

The high-level power analysis flow is depicted in Fig. 1. It requires a high level ArchC model and a register transfer list (RTL) description of the processor. The flow can be organized in two main processes: the characterization process and the simulation process.

The characterization process starts by invoking a synthesis CAD tool, generating a circuit netlist out of the processor RTL-synthesizable hardware description. Typical tools used for this task are Synopsys Design Compiler [20], Xilinx ISE [21] or Altera Quartus [22]. From the circuit netlist it is possible to elaborate a back-annotation with transistor-level power information. A testbench is structured to receive the processor netlist as a device under test, allowing simulations with characterization softwares as inputs. Then, the circuit netlist can be simulated using Mentor Graphics Modelsim [23], generating switching activity files for power analysis tools such

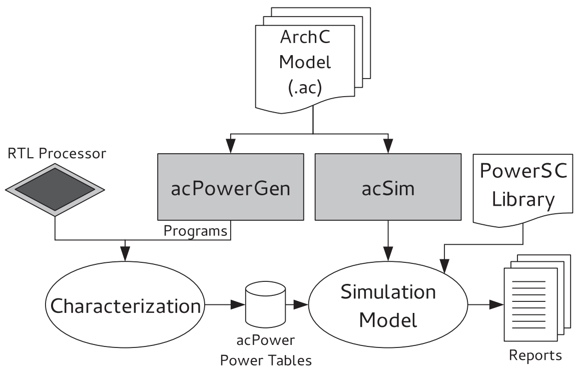


Fig. 1. The acSynth characterization flow report elaboration. The characterization process is detailed in Fig. 2. The simulation model structure is detailed in Fig. 4.

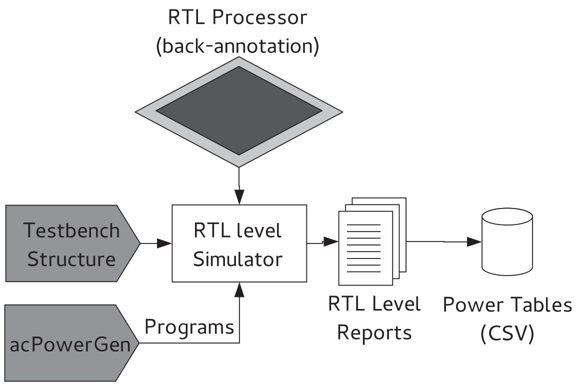


Fig. 2. Characterization process for power information extraction. The RTL processor is synthesized and then, simulated using automatically generated assembly code.

as Xilinx Xpower, Altera PowerPlay or Synopsys RTL Power Estimation flow.

The softwares are automatically generated by our acPowerGen tool. They reproduce the main concepts presented by Tiwari et al. [24,16] when synthesizing the test software. Tiwari shows that the average energy consumption of a characterization program with a loop of several instructions with the same opcode but randomized operands can be used as a power per instruction approximation. In this way, we can obtain a consistent measure of the average power per instruction. By repeating this process to all different instructions, we conclude the energy characterization per instruction and we are able to estimate power using processor frequency and instruction per cycles (IPC) information. This method provides an instruction-based energy consumption analysis and can be used in order to characterize the entire processor instruction set.

The next step, the simulation process, is executed after the RTL power analysis is done. The instruction-based energy consumption information is automatically integrated into the ArchC instruction set simulator by the acSim tool. In this step, the designer uses the power information obtained from the characterization flow into the SystemC simulation. The PowerSC library is the responsible for bringing the data into the model.

The final analysis step consists of the information gathering and report generation. The acSynth final result is a power report and an energy consumption report, enabling a fast power-aware design space exploration and assessment. Each step in Fig. 1 is detailed in the next sections.

3.2. Processor synthesis and power characterization

Fig. 2 provides an overview of the steps to obtain power characterization. We use a CAD RTL suite to generate a back-annotated HDL processor description. Any synthesizer could be used in this step.

Afterwards, the circuit is integrated within a testbench structure and test codes are executed to extract power reports for each processor instruction of interest. We use acPowerGen to generate a set of basic programs to be used in the testbench structure. The acPowerGen tool, described in details in Section 3.3, relies on the assembly language description available on ArchC models to automatically generate these programs. As a result, a set of waveform simulation files is generated, one for each instruction.

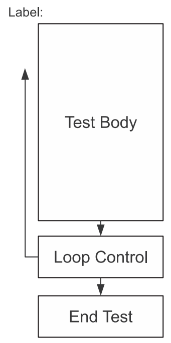
The waveform simulation files are input to power simulation tools to extract power reports. We have used the Xilinx Xpower and Altera PowerPlay tools in our experiments.

3.3. Test code generation

One important step in the characterization process is to automatically generate the set of source code programs to characterize the instruction based energy consumption model. This allows the workflow to be fast and do not rely on designer decisions that may slow down the design space exploration. The test code generation process begins with the ArchC architecture description, which contains information about the processor ISA. For each instruction, the user specifies its behavior using a tree-like semantic description. In that way, acPowerGen knows exactly what each instruction does.

Then, the acPowerGen uses a high-level description of the algorithm that will run on the processor in order to assess the energy consumption of an instruction. This algorithm is based on Tiwari's method, testing the same opcode with different operands. However, it is impractical to generate a large number of instances of the same instruction since it may not fit in the memory module for the simulated processor. Instead, a loop based code structure is generated, similar to Fig. 3a. One sample piece of MIPS-I assembly code is shown in Fig. 3b. These are the first ten lines from instruction `addi` characterization software. The immediate number is randomly generated.

The key challenge to transforming the high-level description of the algorithm lies in the fact that it is not trivial to generate a loop for an unknown instruction set. We overcome this limitation using the semantic information describing each processor instruction in ArchC model. In this way, we gathered sufficient information needed for deducing how a loop may be built and use an



(a) Code generation template

```
globl main
main:
addi $7, $6, -26519
addi $19, $17, 23807
addi $10, $12, 7977
addi $13, $11, 16882
addi $3, $6, 124
addi $2, $20, 10232
addi $8, $7, 17293
addi $22, $3, -15821
(...)
```

(b) First ten lines of an addi characterization software body

Fig. 3. Loop based code structure.

implementation search process to try instructions until we can prove they correspond to the loop, similar to a simple automatic theorem prover used for the ArchC automatic compiler back-end generation [25].

The processor-agnostic algorithm involves allocating a specific machine register to hold the base induction variable and generating instructions to update this register and comparing to check if it met a predefined threshold. In this way, we are able to control how many times the test body is executed.

The last step consists in generating the loop body of the test that contains the instruction to be tested. There is a trade-off between instruction memory consumption versus reduction of the loop control code overhead. We do not want to measure the energy consumption of the loop control instructions, so the larger the number of instructions in the loop body compared to the number of instructions in the loop control, the smaller is this unwanted contribution to the total power measurement.

The loop body contains a single instruction type repeated several times using random operands. The test code generator knows exactly how many operands a given instruction has and their types using the information of the tree-like semantic description. The generator emits the instruction using random operands whether it uses a register number or an immediate value as operand. In this way, we seek to avoid biasing the power measurements to specific operands. All the information used to generate this is available on the ArchC architecture description files and does not require the designer intervention.

3.4. Feedback

The synthesis flow outlined in Section 3.2 generates a processor circuit netlist for a known architecture. Additionally, the characterization process generates reports and data that can now be reused in simulations, saving time and computational resources. Once a processor architecture is described in ArchC, it opens the possibility to use acSim to generate its ISS for simulation and automatic testbench generation. In this way, we can feed characterized data back into the simulation system.

Once data is available, it is possible to use the information into the simulation process. Fig. 4 shows how our framework enables the integration of the ArchC generated simulator with the power information extracted from the previous steps. We used two inputs for it. The first input is the ArchC ISA description files. The second input is the power data generated by the characterization flow, organized as a table file. The acPower power tables used CSV files seeking portability.

The characterization and the simulation processes integration happen when acSim is called using `acpower` parameter, generating a SystemC simulation environment referring PowerSC, acStat and power analysis algorithms.

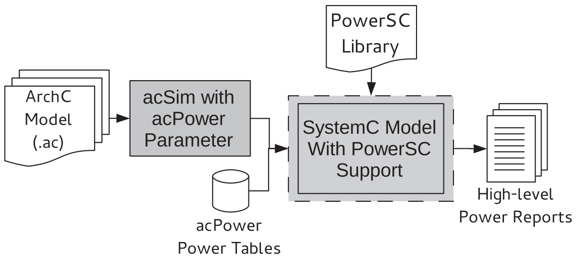


Fig. 4. Power report extraction through SystemC simulation model. The acSynth framework brings power information feedback through acPower, PowerSC and SystemC processor integration.

PowerSC Library. PowerSC is a SystemC extension aiming at the gathering of switching activity [19]. It is a complete framework designed to collect information from any SystemC functional description. As ArchC generates a SystemC processor description, it is an eligible candidate to PowerSC workflow, which abstracts from the user a large share of the process. We have used PowerSC version 0.91.

The acPower Library. The acPower is a library of functions incorpored into ArchC. The acPower was developed to enable the power analysis of ArchC processor modules. Although acPower previous work allowed Tiwari's analysis over some ArchC models, acPower was not fully integrated into the ArchC development flow until its integration discussed here.

We integrated the acPower source code into acStat library. We integrated into the PowerSC core the previous activity collect phase, creating a new tool for power exploration. Moreover, the integrated system shows that other activity-based power analysis algorithms can be easily applied.

The acStat Integrator. The acStat is a key library performing the communication of three fundamental ArchC tools: the acSim ISS generator, the PowerSC library, and the acPower library. The acStat is invoked from the processor simulation environment generated by acSim with acPower parameter. It configures PowerSC statements to collect activity, brings data from CSV external database file into acPower algorithms and then feeds the activity with the algorithms, resulting in energy and/or power reports.

Added to the advantages of the workflow automation, it is now also possible to collect energy consumption versus time, a result not possible using the previous acPower implementation because it was a semi-automatic approach and lacked access to the ISS source code.

3.5. Power model

The inputs to the power model workflow are a synthesis script, and a set of testbenches in Verilog and Assembly language. Each of these parts can be generated without user supervision as explained below.

Synthesis Script The acSynth creates a synthesis script based on the template files previously generated. All file names and parameters are extracted from the baseline ArchC model and we also instantiate the extra components for the testbenches (e.g. a small memory to hold the binary software).

Verilog Testbenches the Verilog testbenches includes instruction memory and data memory and an external execution controller for dynamic instruction characterization purposes.

Assembly Language Programs for a faster power characterization phase, acSynth creates several assembly programs based on the ArchC assembly specification as described in Section 3.3. The instructions fields are fully described and acSynth generates a large set of instructions to enable a good coverage of the behavior value. Each instruction is explored in one different program using the Tiwari's approach to instruction power estimation. Each assembly program has one thousand instructions, all using the operation we are interested in, but with random parameters. The number of instructions was determined empirically, presenting a good number of random parameters restrained in a relative small program. There are two execution modes, normal and forced. In normal mode, the assembly program is executed as it is. In forced mode, an explicit loop control is added into the testbench, executing the code forcefully in sequence and then returning to the first instruction to run again. The forced mode is specially important to simulate branches and jumps where random parameters would generate erratic behavior.

The number of interactions can be easily controlled. However, as expected, there is a threshold where it is worthless going further because it increases processing time without significant changes in estimation. We choose one hundred iterations summing ten thousand instructions per analysis after empirical experiments.

The instruction based power model is created by collecting and computing the power figure for each instruction. The energy of an instruction is determined by the total energy spent in its characterization program divided by the number of occurrences of the instruction in this program. This data is then stored in a CSV file that can be used by acPower in the high level simulator. We augmented acPower to send the energy consumption directly to the PowerSC library so as to collect all energy consumption from a platform.

4. Experimental results

Our experiment's main goal is to assess the effectiveness of the acSynth methodology in different scenarios. Two important inputs guide our power estimation workflow: the subject processor and its implementation technology. The processor architecture, contained in a high level ADL description file, constrains how we will generate the test benchmarks and what instructions we may use. The processor RTL implementation will be used to synthesize a test circuit to the characterization step. Finally, the technology steers how the synthesis tool implements the RTL description into real hardware and thus is a major factor in power estimation.

We test two architectures, MIPS-I and SPARCv8. The underlying microarchitectural implementation of these are the Plasma and the Leon3 core, respectively. In addition to testing two different processors, for the Plasma core we also test two different hardware implementation technologies: Xilinx FPGAs and Altera FPGAs.

To complete the test, after finishing the characterization step in 3 scenarios, we also run Mediabench and Mibench benchmarks, which are too big to run using a traditional power estimation flow, on our fast simulator augmented with data from the characterization step, generating full power reports for real software. For small programs, as the acStone benchmark suite, we compare our simulator with traditional power estimation flow using Altera and Xilinx tools, which operate at the circuit node transition level. This allows us to report the precision of our approach. In all experiments, we only consider the dynamic power because this is the only power factor that is affected by the applications running in the simulations.

4.1. Plasma processor

Plasma is a HDL implementation of the MIPS-I architecture [26]. Its developers maintain a functional server on a Xilinx FPGA platform using the RTOS Plasma operating system to validate its practical application. We choose Plasma because it's an open core project, simple to use and thoroughly validated core. The Plasma CPU has an interrupt controller, UART, memory controller and Ethernet controller. However, we are only interested in the MIPS-I processor core energy consumption and, therefore, in our experiments we used only the mlite_cpu subsystem corresponding to the CPU core. The mlite_cpu includes only the processor pipeline. In this way, we maintain consistence with the usage of this information in the ArchC ISS, which is responsible for accounting solely for the processor core energy consumption.

We generate a back-annotated core description that is then integrated into a testbench infrastructure. Tiwari's method enables the generation of the test software and the final products are waveform files.

4.1.1. Xilinx platform

Table 1 shows the energy consumption characterization of the Plasma instruction set on a Xilinx 3s1200 at 100 MHz. The information in Table 1 is the dynamic energy consumption and therefore it does not account for quiescent power, leakage power or internal power. This static fraction does not depend on the software. Our method aims at capturing how different instruction categories uses different power levels. The key of the method lies in the usage of random operands in the testbenches used to estimate the power of each instruction. As a side effect, Tiwari's method leads to upper bounds in the energy consumption, since in real software the circuit nodes rarely have the amount of transitions provoked by random stimuli.

For example, the MIPS `sw` instruction, responsible for storing a single word to the memory, needs 5.4 nJ on average, considering only the dynamic power, being the highest energy consumption value found for MIPS instructions using Xilinx technologies. It is noteworthy that memory operations are more sensitive to the random operands technique than other regular ALU operations because the memory access datapath uses more power – it involves higher switching activity and communication outside the core module boundaries. Since the characterization table is used to estimate the energy consumption of larger software at fast speeds, this, in turn, leads to an increased overestimation of the energy consumption of regular, non-random, loads and stores that are predominant in software, and it is the cause of larger errors reported in this section. The same was observed in the Altera experiment and will be discussed in Section 4.1.2. However, this is not a random error as addressed in common error theory that uses the normal probability density function to model errors, but it is, instead, a deviation that is generally – but not guaranteed to be – overestimated.

Nevertheless, the `sw` and `lw` energy per instruction information can be better estimated by using a cache memory simulation. Caches could be analyzed with the usage of other tools like CACTI [27] and integrated into the final model. In contrast with these power-hungry instructions, the results show that the `nop` instruction, which is the MIPS *no-operator*, consumes only 0.5 nJ, the lowest energy consumption. We also estimated the stall cycle consumption of 0.206 nJ, which is measured by running the simulation with memory stall control activated.

Using the ArchC ISS augmented with the per instruction power information of Table 1, we measured the energy consumption of the acStone [1], Mibench [28] and Mediabench [29] benchmarks. The average simulation rate was 35 Millions Instructions per Second (MIPS) running on a desktop computer with a 2.66 GHz Intel Core2 Quad Q9450 processor.

Table 1
Dynamic energy consumption of Plasma instructions at 100 MHz.

Instruction	E (nJ)	Instruction	E (nJ)	Instruction	E (nJ)
add	1.865	jr	4.932	sb	4.930
addi	1.791	lb	4.326	sh	5.244
addiu	1.791	lbu	2.130	sll	0.832
addu	1.744	lh	4.480	sllv	0.897
and	0.889	lhu	2.418	slt	1.371
andi	1.051	lui	1.090	slti	1.260
beq	3.977	lw	3.221	sltiu	1.256
bgez	3.977	lwl	3.221	sltu	1.364
bgezal	3.977	lwr	3.221	sra	0.814
bgtz	3.977	mfhi	0.598	srav	0.899
blez	3.977	mflo	0.597	srl	0.810
bltz	3.977	mthi	1.017	srlv	0.891
bltzal	3.977	mtlo	0.949	sub	1.763
bne	3.977	mult	1.457	subu	1.761
div	2.858	multu	1.421	sw	5.427
divu	2.781	nop	0.545	swl	5.427
j	3.412	nor	1.622	swr	5.427
jal	3.805	or	0.977	xor	1.720
jalr	5.027	ori	1.125	xori	1.518

Table 2
Mediabench benchmarks: estimation of energy consumption.

Program	Instr (M)	Power (mW)	Energy (mJ)	t_{Sim} (s)	$\approx t_{Sim}$ RTL (days)
timing	852	175	1489	26	726
rawcaudio	7	174	13	<1	6
rawdaudio	6	181	11	<1	5
toast	221	216	477	7	188
untoast	61	195	119	2	52
cjpeg	17	211	35	1	14
djpeg	5	194	10	<1	4
mpeg2encode	11474	208	23,832	364	9772
mpeg2decode	3772	219	8248	134	3212
pegwit gen	13	181	23	<1	11
pegwit enc	31	182	57	1	27
pegwit dec	17	188	33	1	15

The results for the Mediabench benchmark are presented in Table 2 and for Mibench in Table 3. We tested all inputs of Mibench, but we only show large input cases due to space limitations. For example, in Table 2 we show that the `mpeg2encode` Mediabench program ran the longest instruction trace with 11,474 million instructions, consumed approximately 24 J of energy, the processor operated using 208mW of dynamic power on average, the total simulation time necessary to generate its power estimation was around 6 min and, if this same program were analyzed using the traditional power estimation flow with the XPower tool, we estimate that it would be necessary 9,772 days to complete the power analysis, an impractical solution. We also see that the highest power in Mediabench programs is used when running the `mpeg2decode` program, reaching 219 mW. On the other hand, Table 3 shows that, for Mibench programs, the `crc32` program exceeds 250 mW, demonstrating how the core energy consumption varies with the workload depending on the algorithm, even when using the same processor and technology.

We also performed simulations using 7 parallel applications from the ParMiBench [30] benchmarks. These applications are parallel versions of a subset of Mibench. By evaluating acSynth in parallel benchmarks, we show that the methodology is reproducible for multicore platforms. In Table 4, we present the results for a single core MIPS platform running ParMiBench, and the results for true multicore platforms are reserved for Section 4.3.

The classic RTL workflow time estimations of Tables 2,3 are calculated using the average speed to generate power reports for a reference program, the sieve of Eratosthenes prime number finder, configured to find all prime numbers up to 100. This program executes around 8000 instructions. The Modelsim simulation rate was only 13.3 instructions per second, a difference of six orders of magnitude when comparing to the average ArchC ISS augmented with instruction energy consumption information.

These results show that the default approach using ASIC simulation and power analyzers such as the XPower tool takes much longer under the same conditions and same computational power. On the other hand, the acSynth flow requires a one-time characterization step that takes less than 2 h and a half using the same computational power for simulations. It needs to be done once for each processor-technology. After the characterization process, the ArchC simulation took less than 10 ms to execute the same sieve of Eratosthenes program. In this program, the difference between the power estimation obtained with the traditional flow and the power estimation using the fast acSynth flow was less than 6%, a small error in exchange for an almost instantaneous result.

We also tested the acStone benchmark, presented in Table 6.[1] The acStone benchmark has a collection of testing software and it can be downloaded from the ArchC site [1]. This benchmark is geared at validating processor designs using small programs. Since they are usually very small, we used acStone to estimate power using not only the acSynth methodology, but also using the standard RTL power estimation flow with the XPower tool. Hence, instead of estimating the RTL simulation time as in the Mediabench and Mibench benchmarks, for acStone it was possible to provide real values for the simulation times of the XPower approach and also present the difference of the energy consumption reported by acSynth versus the XPower.

Since our characterization process is based on a RTL simulation of our characterization software, when using the per-instruction fast estimation we lose precision in comparison with the RTL approach and thus we say that the difference between the powers reported by acSynth and by XPower is our error. For example, Table 6 shows that the `015.const` program executed 37 instructions, the acSynth workflow measured the core power as 135.85 mW while XPower measured it to be 157.61 mW and thus the error was 13.81%. It is possible to see that, as we argued earlier that acSynth usually overestimates, the majority of power report numbers from acSynth are larger than those from XPower.

In early simulations, the largest errors appear in multiplication and division benchmarks. The reason is that these benchmarks suffered a large number of stall cycles due to data hazards, events that are ignored by an ISS fast simulator. When running the characterization programs of `mult` and `div`, there are no such data hazards, since the MIPS `lo` and `hi` registers cannot be used as direct operands of other `mult` or `div` instructions. Without data hazards, the pipeline executes without pause, with *increased power*, while the presence of data hazards slows down the pipeline, *lowering power*.

When tallying total energy consumption in a program by using a fixed value of energy consumption for long arithmetic instructions such as multiplication and division, we observe two effects: first, total energy consumption is underestimated, since real programs often have stalls, which accounts for extra cycles using energy while idling, and second, the final average power is overestimated, since real programs with stalls introduce idle cycles with low activity, which reduces energy consumption per unit of time. For example, acSynth estimated the power used in `051.mul` to be 251.14 mW because that would make sense if all computation was done without interruption at a high pace. This leads to the error of 145.54% between acSim and Modelsim simulations for 051.mul.

For these special cases, to explain the nature of the discondance between our simulation and the RTL gate-level model, we performed additional simulations considering `mult` and `div` instructions as macros to be expanded as multiple instructions in our simulator: the instruction itself and additional "stall" instructions, accounting for the additional cycles with stall energy. In this approach, the error dropped considerably compared to previous

[1] We showed only a few programs here due to space restrictions. All acStone benchmark applications were executed.

Table 3
Mibench benchmarks: estimation of energy consumption.

Program	Instr (M)	Power (mW)	Energy (mJ)	t_{Sim} (s)	$\approx t_{Sim}$ RTL (days)
Consumer					
lame	94,315	207.68	195,872	3047	80,324
cjpeg	109	209.74	229	3	93
djpeg	29	207.54	61	1	25
Automotive					
basicmath	22,269	208.70	46,476	765	18,966
bitcnts	684	195.69	1339	19	583
qsort	989	216.12	2138	31	843
susan cor	45	203.90	91	1	38
susan edg	177	187.96	333	5	151
susan smo	423	155.55	659	12	361
Network					
dijkstra	285	192.36	549	8	243
patricia	1831	211.86	3879	65	1559
Office					
search	7	244.05	17	<1	6
Security					
rijndael dec	361	208.34	752	12	308
rijndael enc	351	206.88	726	12	299
sha	136	181.73	247	4	116
Telecommunication					
adpcm rawc	689	172.87	1191	24	587
adpcm rawd	539	179.06	965	20	459
adpcm tim	688	174.76	1203	25	586
crc_32	615	250.60	1541	22	524
fft 32k	15,244	205.19	31,280	534	12,983
fft inv 32k	14,750	204.74	30,201	474	12,562
gsm toast	1763	216.22	3813	58	1502
gsm unt	523	199.27	1043	17	446

Table 4
ParMiBench benchmarks: estimation of energy consumption in single-MIPS platforms.

Program	Instr (M)	Power (mW)	Energy (mJ)	t_{Sim} (s)	$\approx t_{Sim}$ RTL (days)
basicmath	5	220	303	<2	4
dijkstra	5	179	234	<2	4
fft	92	205	5330	26	78
sha	1	212	64	<1	<1
stringsearch	116	208	7072	34	98
susan-corners	83	223	5798	26	70
susan-edges	198	199	11,343	57	168
susan-smoothing	93	233	6990	30	79

Table 5
Simulation with *acStone* and error evaluation (Plasma Altera).

Program	Instr	acSynth (mW)	PowerPlay (mW)	Error (%)
018.const	56	63.20	57.98	9.01
031.add	259	66.89	65.69	1.83
051.mul	96	66.05	65.88	0.27
061.div	192	69.77	67.33	3.62
071.bool	281	65.45	67.77	3.42
111.if	318	69.12	66.65	3.71
121.loop	959	70.28	65.67	8.55
132.call	271	69.07	64.41	7.24
142.array	94,771	62.50	65.74	4.93

Table 6
Simulation with *acStone* and error evaluation (Plasma Xilinx).

Program	Instr	acSynth (mW)	Xpower (mW)	Error (%)
011.const	34	131.45	146.77	10.44
021.cast	44	185.59	186.75	0.62
031.add	259	263.80	240.71	9.59
041.sub	259	265.18	243.22	9.03
051.mul*	96	101.19	102.28	1.86
061.div*	192	124.10	107.52	5.94
071.bool	281	266.90	240.51	10.97
081.shift	183	282.18	235.6	19.77
111.if	318	260.25	228.4	13.94
112.if	318	231.12	222.97	3.66
121.loop	959	256.30	223.72	14.56
131.call	111	242.97	255.9	5.05
142.array	94,771	168.39	128.71	30.83

results. Previously, `051.mul` executed 96 instructions, but with this approach the acSim simulation tallies 342 instructions, many of them being stalls. Although the simulation still reports a similar amount of energy spent, the estimated power is lower due to the larger denominator (number of instructions). In this scenario, the error for `051.mul` is only 1.86%. The same was observed in all acStone simulations with high use of `mult` and `div` instructions. In Table 6 the special cases are marked with an asterisk. This strategy can be used in the final model of Plasma to preserve simulation speed, although a straightforward solution would be to enhance the ArchC framework with cycle-accurate information, at the cost of slowing down simulation.

On the other hand, while long delay instructions may cause noticeable distortions due to long-standing data hazards, it is important to note that arithmetic instructions usually account for only a fraction of the total instructions of a program. AcStone is not composed of real programs, but small tests built to repeat the execution of a specific class of instructions. In real benchmarks, the use of multiplication and division instructions is much more sparse and using a single value of energy for all multiplication and division instructions is enough to estimate power reasonably, even for Plasma in this Xilinx platform. We evaluate this hypothesis by comparing the results of RTL Modelsim simulation of a reduced Mediabench cjpeg benchmark and the acSynth simulation environment. Since the RTL simulation takes too long, we reduced the size of the input image to a small 50×33 pixels BMP image, making the simulation time feasible. The number of executed instructions was less than 700,000. The simulation reported the power to be 240 mW. The same software was used in acSynth framework, without changing the `mult` and `div` to macros as in the previous strategy. Even simulating in the presence of the characterization problem discussed before, the error was much smaller. The ArchC simulation reported 252 mW, a relative error of 15.68%.

For other platforms, as we will present shortly, this problem is less intense.

Excluding the special situation presented by multiplication and division, the analysis with MIPS-I and Xilinx tool set presented error between 0.02% and 61.05%, with 91% of the total number of analyzed cases presenting errors with less than or equal to 30%.

Fig. 5 presents instantaneous powers graphics for Mediabench and Mibench benchmarks. Besides computing energy consumption and final power estimates, the acSynth also produces instantaneous power graphics, which help the designer understanding the correlation between algorithms and the energy consumption.

4.1.2. Altera platform

We performed the same experiments with a Plasma processor in an Altera Cyclone-V 5CGXFC7C7F23C8 FPGA at 25 MHz. The development flow was the same as the previous process presented for Xilinx, with exception of the configuration of an acSynth file to use the Altera tools, Quartus and PowerPlay, instead of the Xilinx tools. Similarly, the Plasma code needed to be refactored to use memories compatible with the Altera FPGA family.

The simulations for the Altera characterization step were performed at the HDL level. The characterization of the system consumed approximately one hour. For Altera, we estimated stall cycles to consume 1.64 nJ (versus 0.206 nJ of Xilinx), which is measured by running the simulation with memory stall control activated. The characterization of dynamic energy consumption are not presented here due to space limitations. In the limits, `j` energy was characterized with 3.93 nJ and NOP with 1.94 nJ.

We included the per instruction power information into the ArchC ISS and we measured the energy consumption of ArchC benchmarks. For Mibench, all the tests were made but only the results for large inputs are presented due to space limitations. The estimate of RTL simulation time was equally high. As expected, acSynth still keeps a large gain in simulation time for a different platform and a different workflow.

Both `load` and `store` instructions presented high-energy results again. The instruction `lw` shows 2.98 nJ of energy-per-instruction and `sw`, 3.56 nJ. We can also notice that branches instructions consume more energy due to higher switching activities over input and output pins, suggesting that there is a high consumption associated with input and output pads. Considering that Tiwari's method has the tendency of overestimation, we can expect bigger errors when external data access (outside core) is involved. Integrating the cache power analysis from tools such as CACTI could reduce the deviation in those cases.

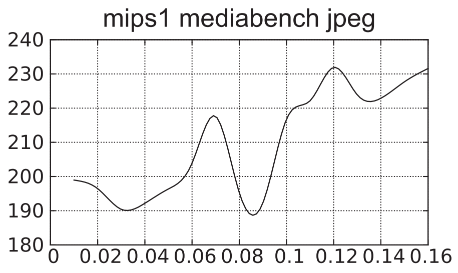
We also tested the acStone benchmark on the Altera platform. Partial results are presented in Table 5 due to space limitations. We can see an overall reduction in error. The error range here was between 0.01% and 17.49%, with 96% of the total number of analyzed cases showing error with less than or equal to 15%. The multiplication and division issue was considerably reduced here. We reckon that the Plasma layout for this Altera FPGA was synthesized to clock at a lower frequency, 25 MHz, increasing memory access slack time and, therefore, reducing stall occurrences.

4.2. Leon3 processor

Leon3 is a HDL implementation of the SPARCv8 architecture developed by *Gaisler Research* [31]. The processor is distributed under two different licenses, a LGPL license and a commercial one. The LGPL license allows access to the source code and free use for academic purposes. We chose Leon3 because it is compatible with the SPARCv8 ArchC processor model and, since it is more complex than Plasma, the evaluation of the acSynth workflow on this platform reveals different aspects. In contrast with the 100 MHz FPGA of Plasma, Leon3 was characterized on a Xilinx Spartan3 xc3s1000 FPGA clocked at 50 MHz.

The complexity of Leon3 affected the size of transition files due to the larger number of signals and registers. As a result, there was an increase in the time spent with characterization. Each instruction spent 5-10 min to collect the transition activities and 10–15 min to generate the XPower reports. We performed 97 tests, tallying up to 30 h of characterization. After the automatic characterization, some instructions were filled manually using a counterpart instruction: `load`, `store` and `swap` with register operands were filled with energy consumption of the same instruction using immediate values; the energy of the division instructions were filled with the energy consumption of their equivalent multiplication instructions; the `jump`, `jmpl_imm`, and `jmpl_reg` were filled with the energy consumption of the `ba` instruction. The power consumption of the `stall` cycle was estimated to be 84 pJ. In Table 7 we present the final report.

It is noteworthy that the Leon3 energy consumption per instruction uses a pico-Joule scale, presenting a lower consumption than Plasma. We may credit this to the different chip characteristics



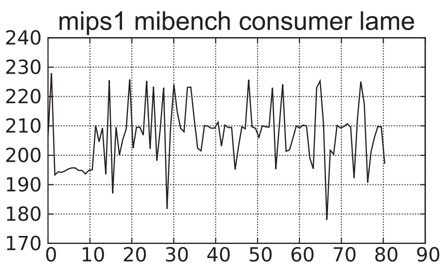


Fig. 5. Energy profile of the Mediabench and Mibench benchmarks using characterization data on Xilinx Platform. Power (mW) $\times$ Time (s).

of this FPGA. Using the ArchC SPARCv8 ISS augmented with the per instruction power information of Table 7, we measured the energy consumption of Mibench and Mediabench benchmarks. Table 8 presents the results for the Mediabench benchmark.

We also carry out a comparative study between the RTL level simulation and the ADL level simulation using the acStone benchmark, whose results are shown in Table 9. In comparison with Plasma, note the difference in the number of executed instructions: the program 146.array required 29,248 MIPS instructions on Plasma while on Leon3 the same computation finishes after 21,288 SPARC instruction are executed. This difference between architectures can be especially important when energy consumption is a concern. If the algorithm is taken into consideration in the consumption analysis, it can influence which processor a developer should choose. The sole analysis of average power consumption can lead to mistakes if you do not consider the amount of instructions required to execute a program over all candidates.

It is important to notice that we applied the macro expansion approach in this simulation for softwares abusing `mul` and `div`. As discussed in Section 4.1.1, `mult` and `div` may increase the error due to stalls. For SPARC, we applied the same type of instruction count adjustment as follows: multiplication instructions were expanded to 5 (to simulate the 5 cycles of delay in a 16×16 pipelined multiplier), and division instructions to 36. As expected, the distortion decreased in those cases. These special cases are marked with an asterisk in Table 9. The Leon3 processor is more complex than the Plasma processor and this fact also has an impact on the technique. The adjustment brought better results, but not as good as in the Plasma processor case study. The reason is that Leon3 has more complex scenarios and stall cycles appears in both characterization and simulation steps. And as we discussed for MIPS Plasma, a straightforward solution would be to enhance the ArchC framework with cycle-accurate information.

For the SPARCv8 architecture using Xilinx tools, the error ranged between 0.14% and 40.66% with 95% of the total number of cases presenting errors less than or equal to 20%. The greater error was, like Plasma for Xilinx, in multiplication and division benchmarks, although it was smaller than for Plasma. This suggests that Leon3 has a more consistent mult/div unit co-processor and that the lower frequency (50 MHz, compared to Plasma 100 MHz at Xilinx platform) can affect the error rate.

Table 7
Dynamic energy of Leon3 instructions in a Spartan3 FPGA.

Instruction	E (pJ)	Instruction	E (pJ)	Instruction	E (pJ)
add_imm	272.00	ldsb_reg	118.20	st_imm	189.40
add_reg	396.00	ldsh_imm	117.00	st_reg	189.40
addcc_imm	286.00	ldsh_reg	117.00	stb_imm	229.80
addcc_reg	396.80	ldstub_imm	149.60	stb_reg	235.20
addx_imm	282.20	ldstub_reg	149.60	std_imm	167.20
addx_reg	394.60	ldub_imm	111.00	std_reg	167.20
addxcc_imm	282.80	ldub_reg	111.00	sth_imm	220.40
addxcc_reg	397.40	lduh_imm	111.40	sth_reg	220.40
and_imm	188.20	lduh_reg	111.40	sub_imm	298.60
and_reg	174.20	mulscc_imm	256.40	sub_reg	405.20
andcc_imm	191.00	mulscc_reg	278.80	subcc_imm	302.20
andcc_reg	175.00	nop	149.60	subcc_reg	407.80
andn_imm	205.60	or_imm	217.60	subx_imm	299.40
andn_reg	176.20	or_reg	182.60	subx_reg	405.20
andncc_imm	209.80	orcc_imm	217.00	subxcc_imm	300.00
andncc_reg	177.40	orcc_reg	183.00	subxcc_reg	407.40
ba	102.80	orn_imm	222.20	swap_imm	137.60
bcc	106.20	orn_reg	193.00	swap_reg	137.60
bcs	151.60	orncc_imm	224.60	trap_imm	0.00
be	102.40	orncc_reg	194.40	trap_reg	0.00
bg	151.80	rdy	160.00	udiv_imm	133.20
bge	106.40	restore_imm	170.00	udiv_reg	115.00
bgu	151.80	restore_reg	169.60	udivcc_imm	144.40
bl	151.80	save_imm	171.00	udivcc_reg	113.60
ble	102.40	save_reg	171.40	umul_imm	133.20
bleu	102.20	sdiv_imm	171.00	umul_reg	115.00
bn	150.00	sdiv_reg	116.00	umulcc_imm	144.40
bne	150.40	sdivcc_imm	144.40	umulcc_reg	113.60
bneg	151.60	sdivcc_reg	115.20	unimplemented	0.00
bpos	106.60	sethi	244.80	wry_imm	324.00
bvc	106.40	sll_imm	195.20	wry_reg	218.80
bvs	151.80	sll_reg	183.40	xnor_imm	345.40
call	103.80	smul_imm	171.00	xnor_reg	375.60
jmpl_imm	102.80	smul_reg	116.00	xnorcc_imm	347.20
jmpl_reg	102.80	smulcc_imm	167.60	xnorcc_reg	380.00
ld_imm	111.00	smulcc_reg	115.20	xor_imm	257.00
ld_reg	111.00	sra_imm	190.60	xor_reg	288.00
ldd_imm	126.40	sra_reg	176.20	xorcc_imm	265.40
ldd_reg	169.80	srl_imm	193.60	xorcc_reg	281.60
ldsb_imm	118.20	srl_reg	175.80		

Table 8
Mediabench benchmarks: estimation of energy consumption.

Program	Instr (M)	Power (mW)	Energy (mJ)	t_{Sim} (s)	$\approx t_{Sim}$ RTL (day)
timing	868	12	208	27	739
rawcaudio	7	12	2	<1	6
rawdaudio	6	11	1	<1	5
toast	157	11	35	5	134
untoast	85	12	21	3	72
cjpeg	14	12	3	<1	12
djpeg	4	12	1	<1	4
mpeg2encode	10,834	11	2299	380	9227
mpeg2decode	3452	10	664	123	2940
pegwit gen	13	12	3	<1	11
pegwit enc	30	11	7	1	26
pegwit dec	17	11	4	1	14

4.3. Energy profiling of multicore systems

As a beneficial side effect, acSynth can provide the energy profile in real-time. Using the acStat class, it is possible to create different algorithms for power analysis. During the simulation, acSynth saves the data in CSV format with instantaneous power estimation. This feature allows us to plot graphs offering a different perspective into power analysis with special focus on the relationship between software and processor in the energy consumption.

Hitherto, we presented the energy profile for a single processor core. The framework may be extended using PowerSC to generate whole system reports encompassing several modules in different abstraction levels. For instance, one could have a processor at the architecture level, memory at system level, and accelerators at gate

Table 9
Simulation with *acStone* and effective error evaluation (Leon3 Xilinx).

Program	Instr	acSynth (mW)	XPower (mW)	err_{eff} (%)
011.const	31	9.13	8.72	4.67
021.cast	45	9.10	8.56	6.29
031.add	222	10.73	9.56	12.25
041.sub	222	10.78	9.48	13.73
051.mul*	381	7.28	8.80	16.76
052.mul*	282	7.80	9.15	14.36
061.div*	769	9.84	9.46	3.66
071.bool	238	10.09	9.12	10.60
081.shift	186	10.27	9.42	9.00
111.if	381	9.51	8.79	8.21
121.loop	703	9.57	8.51	12.51
125.loop	1489	8.57	8.56	0.14
131.call	94	8.52	8.50	0.28
146.array	21,288	9.06	10.04	9.80

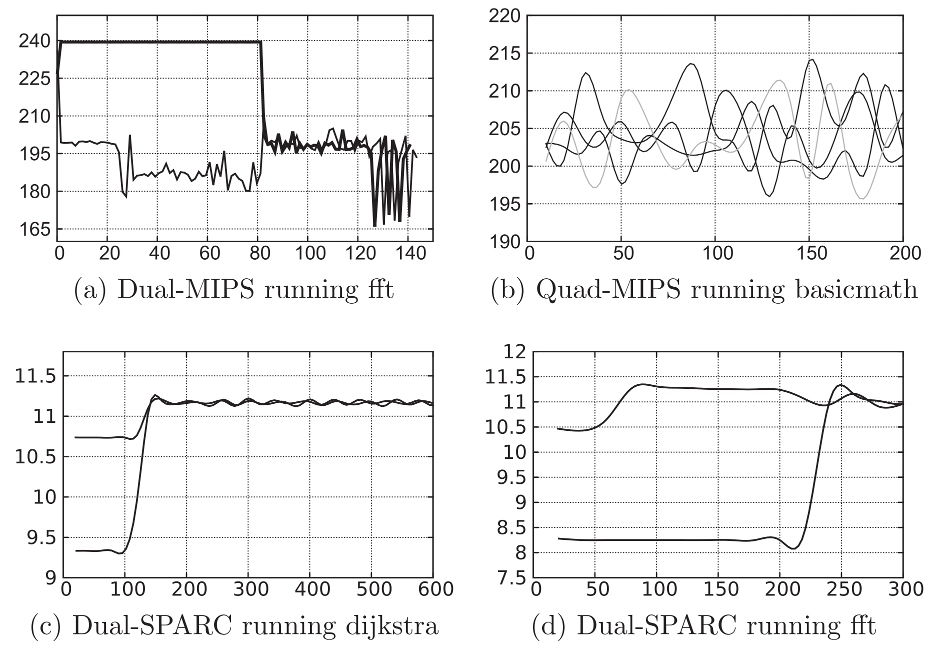


Fig. 6. Energy profile of SPARC and MIPS platforms [Power (mW) × Time (s)].

level. Furthermore, each module can have a specific analysis algorithm, which is an appealing flexibility to the system designer.

We found that acSynth and its methodology are easily applied to virtual multicore platforms. To demonstrate this, we use a scalable set of SystemC-based representation of MultiPlocessor SoCs (MPSoCs), each one composed by 1, 2, 4, 8, 16, 32, or 64 cores, a shared memory, a lock device and a router intercommunication device. These platforms use the ArchC MIPS-I and SPARCv8 processor models, including the per instruction power information per core provided by acSynth.

The acSynth reports show energy consumption details, which can be used for power profiling as shown in Fig. 6. The elementary difference between energy consumption of the two cores in Fig. 6a–d is caused by the larger number of memory accesses to allocate and initialize the input variables; this setup phase is executed by one of the cores while the other is waiting in a barrier. In this case, we can identify different phases performed by each core. A further observation is that the energy consumption of the cores in Fig. 6b reflects a large number of mathematical operations carried out over a small statically allocated amount of data.

5. Conclusion

This paper presented acSynth, an ArchC framework for power characterization and simulation. This tool brings a whole new consumption analysis aspect into ArchC allowing power reports and energy consumption to be generated in a very short time frame. As main contributions, we can assign:

- we introduced full and expansible characterization methodology, independent of synthesis tools and target architecture;
- we illustrated how to generate the characterization programs using ADL information, and we detailed the integration of PowerSC, acPower and acSim tools elaborating a unified system bringing power consumption analysis into the ArchC ADL;
- we presented the acSynth tools, applying our methodology over two processors, Plasma and Leon3, with distinguished architectures, MIPS-I and SPARCv8, developed over two independent workflow platforms, Altera and Xilinx;
- we showed a systematic method to expand acSim in order to bring new high level analysis into the ArchC simulator, as power consumption reports, and also integrate this method to extract power information from multicore simulations using ArchC processors.

Our experiments with the framework demonstrate that the power characterization of Plasma, a MIPS-I processor, could be performed in roughly 2 h with very small accuracy diversion. The same characterization would take years on standard RTL methodologies. The short time provided by the framework is possible due to the fast simulation speeds (35 millions instructions per second), achieved on a regular quad-core desktop computer. The acSynth brings a powerful architecture design level power analyzer applicable to any architecture supported by ArchC system and community.

The analysis with MIPS-I and Xilinx tool set presented error between 0.02% and 61.05%, with 91% of the total number of analyzed cases presenting errors with less than or equal to 30%. Adopting Altera tool set, the error was between 0.01% and 17.49% with 96% of the total number of analyzed cases showing error with less than or equal to 15%. For SPARCv8 architecture, using Xilinx tool set, the error ranged between 0.14% and 40.66% with 95% of the total number of analyzed cases presenting errors with less than or equal to 20%. The methodology was also successfully applied in a multicore simulation system, providing power profiles and comparative simulations at the architecture level.

Acknowledgments

This work was partially funded by FAPESP grants 2010/02230-5, 2011/09630-1, 2011/00901-2, CAPES and CNPq.

References

[1] The ArchC Architecture Description Language. Available at: <http://www.archc.org/>.

[2] S. Rigo, G. Araujo, M. Bartholomeu, R. Azevedo, Archc: a systemC-based architecture description language, in: 16th Symposium on Computer Architecture and High Performance Computing, 2004 – SBAC-PAD 2004,

2004, pp. 66–73 (Best Paper Award). http://dx.doi.org/10.1109/SBAC-PAD.2004.8.
[3] R. Azevedo, S. Rigo, M. Bartholomeu, G. Araujo, C. Araujo, E. Barros, The ArchC architecture description language and tools, International Journal of Parallel Programming 33 (2005) 453–484, http://dx.doi.org/10.1007/s10766-005-7301-0.
[4] S. Rigo, R. Azevedo, L. Santos, Electronic System Level Design: An Open-Source Approach, Springer, 2011.
[5] T. Gupta, C. Bertolini, O. Heron, N. Ventroux, T. Zimmer, F. Marc, High level power and energy exploration using ArchC, in: 22nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), 2010, 2010, pp. 25–32. http://dx.doi.org/10.1109/SBAC-PAD.2010.13.
[6] J. Ma, R. Azevedo, Estimativa de consumo de energia em nivel de instrucao para processadores modelados em ArchC, in: Workshop de Sistemas Computacionais – WSCAD-SSC, SBC, 2009, pp. 119–126 (in Portuguese).
[7] M. Guedes, R. Auler, E. Borin, R. Azevedo, An ArchC approach for automatic energy consumption characterization of processors, in: Proceedings of the 23rd IEEE International Symposium on Rapid System Prototyping, RSP 2012, 2012.
[8] J. Coburn, S. Ravi, A. Raghunathan, Power emulation: a new paradigm for power estimation, in: Proceedings of the 42nd Annual Design Automation Conference, DAC '05, ACM, New York, NY, USA, 2005, pp. 700–705. http://dx.doi.org/10.1145/1065579.1065764. Available at: <http://doi.acm.org/10.1145/1065579.1065764>.
[9] T.H. Krodel, Powerplay-fast dynamic power estimation based on logic simulation, in: Proceedings of the 1991 IEEE International Conference on Computer Design on VLSI in Computer & Processors, ICCD '91, IEEE Computer Society, Washington, DC, USA, 1991, pp. 96–100. Available at: <http://dl.acm.org/citation.cfm?id=645460.654390>.
[10] W. Ye, N. Vijaykrishnan, M. Kandemir, M.J. Irwin, The design and use of simplepower: a cycle-accurate energy estimation tool, in: Proceedings of the 37th Annual Design Automation Conference, DAC '00, ACM, New York, NY, USA, 2000, pp. 340–345. http://dx.doi.org/10.1145/337292.337436. Available at: <http://doi.acm.org/10.1145/337292.337436>.
[11] E. Senn, J. Laurent, N. Julien, E. Martin, Softexplorer: estimating and optimizing the power and energy consumption of a C program for DSP applications, EURASIP Journal on Applied Signal Processing 2005 (2005) 2641–2654, http://dx.doi.org/10.1155/ASP.2005.2641. Available at: <http://dx.doi.org/10.1155/ASP.2005.2641>..
[12] A. Stammermann, L. Kruse, W. Nebel, A. Pratsch, E. Schmidt, M. Schulte, A. Schulz, System level optimization and design space exploration for low power, in: Proceedings of the 14th International Symposium on Systems Synthesis, ISSS '01, ACM, New York, NY, USA, 2001, pp. 142–146. Available at: <http://dx.doi.org/10.1145/500001.500034>.
[13] O. Schliebusch, A. Chattopadhyay, R. Leupers, G. Ascheid, H. Meyr, M. Steinert, G. Braun, A. Nohl, Rtl processor synthesis for architecture exploration and implementation, in: Proceedings of the Conference on Design, Automation and Test in Europe – Volume 3, DATE '04, IEEE Computer Society, Washington, DC, USA, 2004, p. 30156.
[14] L. Benini, D. Bertozzi, A. Bogliolo, F. Menichelli, M. Olivieri, Mparm: Exploring the multi-processor SoC design space with systemC, The Journal of VLSI Signal Processing 41 (2005) 169–182, http://dx.doi.org/10.1007/s11265-005-6648-1.
[15] L. Piga, R. Bergamaschi, F. Klein, R. Azevedo, S. Rigo, Empirical web server power modeling and characterization, IEEE International Symposium on Workload Characterization (IISWC) 2011 (2011) 75, http://dx.doi.org/10.1109/IISWC.2011.6114200.
[16] V. Tiwari, S. Malik, A. Wolfe, M.T. chien Lee, Instruction level power analysis and optimization of software, Journal of VLSI Signal Processing 13 (1996) 1–18.
[17] V. Degalahal, T. Tuan, Methodology for high level estimation of FPGA power consumption, in: Design Automation Conference, 2005. Proceedings of the ASP-DAC 2005. Asia and South Pacific, vol. 1, 2005, pp. 657–660. http://dx.doi.org/10.1109/ASPDAC.2005.1466245.
[18] M. Guedes, R. Auler, E. Borin, R. Azevedo, An ArchC approach for automatic energy consumption characterization of processors, in: Rapid System Prototyping (RSP), 2012, 2012.
[19] F. Klein, G. Araujo, R. Azevedo, R. Leao, dos Luiz Santos, An efficient framework for high-level power exploration, in: MWSCAS 2007: Proceedings of the 50th Midwest Symposium on Circuits and Systems, 2007, pp. 1046–1049. http://dx.doi.org/10.1109/MWSCAS.2007.4488741.
[20] Synopsys Tools. Available at: <http://www.synopsys.com/Tools/>.
[21] Xilinx Design Tools. Available at: <http://www.xilinx.com/products/design-tools/>.
[22] Altera Design Software. Available at: <http://www.altera.com/products/software/>.
[23] Modelsim – Advanced Simulation and Debugging. Available at: <http://model.com/>.
[24] V. Tiwari, S. Malik, A. Wolfe, Power analysis of embedded software: a first step towards software power minimization, IEEE Transactions on VLSI Systems 2 (1994) 437–445.
[25] R. Auler, P. Centoducatte, E. Borin, ACCGen: an automatic ArchC compiler generator, in: Proceedings of the IEEE 24th International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD '12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 278–285. http://dx.doi.org/10.1109/SBAC-PAD.2012.33.
[26] S. Rhoads, Plasma-most MIPS I (TM) Opcodes: Overview. Available at: <http://opencores.org/project> (02.05.12).
[27] P. Shivakumar, N.P. Jouppi, Cacti 3.0: An Integrated Cache Timing, Power, and Area Model, 2001/2.
[28] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, R.B. Brown, Mibench: a free, commercially representative embedded benchmark suite, in: Proceedings of the Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop, WWC '01, IEEE Computer Society, Washington, DC, USA, 2001, pp. 3–14. http://dx.doi.org/10.1109/WWC.2001.15.
[29] C. Lee, M. Potkonjak, W.H. Mangione-Smith, MediaBench: a tool for evaluating and synthesizing multimedia and communications systems, in: Proceedings of the 30th Annual ACM/IEEE international Symposium on Microarchitecture, MICRO 30, Washington, DC, USA, 1997, pp. 330–335.
[30] S.M.Z. Iqbal, Y. Liang, H. Grahn, ParMiBench: an open-source benchmark of embedded multiprocessor systems, IEEE Computer Architecture Letters 9 (2) (2010) 45–48.
[31] A. Jiri Gaisler, Aeroflex Gaisler. Available at: <http://www.gaisler.com/> (Oct 2012).

Marcelo Guedes received his Master degree in computer science from University of Campinas (UNICAMP), Sao Paulo, Brazil, in 2012. He currently works as a circuit designer in Idea-IP.

Rafael Auler received his Master degree in computer science from University of Campinas (UNICAMP), Sao Paulo, Brazil, in 2011. He currently is enrolled in his Ph.D. at the same university.

Liana Duenha received her Master degree in computer science from Federal University of Mato Grosso do Sul (UFMS), Mato Grosso do Sul, Brazil, in 2002. She currently is enrolled in her Ph.D. at University of Campinas – UNICAMP.

Edson Borin received his Ph.D. degree in computer science from University of Campinas (UNICAMP), Sao Paulo, Brazil, in 2007. He currently works as a Professor with the Institute of Computing at UNICAMP. Prior he worked at Intel Labs, California, where he published 7 patents and got 4 internal prizes.

Rodolfo Azevedo received the Ph.D. degree in computer science from University of Campinas (UNICAMP), Sao Paulo, Brazil, in 2002. He currently works as a Professor with the Institute of Computing at UNICAMP. His main research interests include computer architecture, low power, memory technologies, ADLs, and system-level design, mainly using SystemC. Dr. Azevedo was a corecipient of Best Paper Awards at SBAC-PAD'04 for his work on the ArchC ADL and SBAC-PAD'08.