CREEP: Chalmers RTL-based Energy Evaluation of Pipelines

Daniel Moreau[†], Alen Bardizbanyan[†], Magnus Själander[‡], and Per Larsson-Edefors[†]

[†]Chalmers University of Technology, Gothenburg, Sweden
[‡]Norwegian University of Science and Technology, Trondheim, Norway

January 10, 2017
Technical report 2017:01, ISSN 1652-926X
Dept. of Computer Science and Engineering, Chalmers University of Technology

***Abstract*—Energy estimation at architectural level is vital since early design decisions have the greatest impact on the final implementation of an electronic system. It is, however, a particular challenge to perform energy evaluations for processors: While the software presents the processor designer with methodological problems related to, e.g., choice of benchmarks, technology scaling has made implementation properties depend strongly on, e.g., different circuit optimizations such as those used during timing closure. However tempting it is to modularize the hardware, this common method of using decoupled pipeline building blocks for energy estimation is bound to neglect implementation and integration aspects that are increasingly important. We introduce CREEP, an energy-evaluation framework for processor pipelines, which at its core has an accurate 65-nm CMOS implementation model of different configurations of a MIPS-I-like pipeline including level-1 caches. While CREEP by default uses already existing estimated post-layout data, it is also possible for an advanced user to modify the pipeline RTL code or retarget the RTL code to a different process technology. We describe the CREEP evaluation flow, the components and tools used, and demonstrate the framework by analyzing a few different processor configurations in terms of energy and performance.**

I. Introduction

In the early days of IC design, processors were developed with a focus on achieving high performance. Other design factors such as cost, area and power dissipation were also considered but only as limiting factors. However, in the late 1990's it became apparent that this design philosophy was unsustainable. CMOS technology scaling allowed for higher densities and increasing clock rates, but performance-centered designs that tried to leverage these advances became hard or impossible to cool in a cost-effective manner [1]. The power wall, which is a direct consequence of a discontinued Dennard scaling, means that technology scaling no longer is the obvious answer to increased performance and lower power [2], [3].

Energy efficiency is, next to performance, the major focal point in VLSI design. The driving forces behind this are increased portability and environmental concerns. For portable battery-powered devices lower energy dissipation directly translates into a more well-received product. As far as environmental concerns, it is becoming painfully obvious that the rate at which the global energy dissipation increases is not sustainable. What is worrying is that integrated circuits contribute to a considerable chunk of this increase [4].

To facilitate energy-efficient design, architectural evaluation frameworks are required to enable vital early estimations to allow for more predictable prototyping results [5]. Early estimations are perhaps the most important estimations as changes at the architecture level have a larger impact on the final energy and performance numbers than changes at the circuit level. Since such estimation frameworks are significantly faster than those available at the circuit level, workloads that are impractical (or even impossible) to use during register transfer level (RTL) power estimation become feasible to evaluate. However, these frameworks have traded speed for accuracy by neglecting the actual circuit integration, by adopting parameterizable models that are obtained through analytical or empirical studies of underlying hardware components [6]–[10]. Since current frameworks tend to neglect the impact of implementation, the synergy between the integrated parts of a design is not considered.

We introduce the open-source CREEP framework, *Chalmers RTL-based Energy Evaluation of Pipelines* [11], that models a MIPS-I-like pipeline that is fully integrated with level-1 (L1) cache memories. Since the CREEP framework extends down to RTL and place-and-route, it yields high accuracy and allows for detailed pipeline studies at the system level. Furthermore, the RTL provided in CREEP is written in a modular fashion and can easily accommodate different pipeline extensions. The operation of CREEP is automated through scripts to make the framework user-friendly and, more importantly, results fully reproducible from one evaluation round to another. The open-source package of CREEP supports several configurations; in particular different configurations of caches which have a very significant impact on the processor performance and energy.

This report is organized as follows: Sec. II gives an overview of CREEP and briefly reviews the microarchitecture assumed and the architecture simulator used. In Sec. III, we present the components of the CREEP framework as they are applied in an evaluation flow. Sec. IV gives a brief introduction to the practical handling of CREEP, while Sec. V provides a demonstration of the capability of CREEP as it is used to evaluate different processor configurations with respect to performance as well as power and energy dissipation.

II. Introduction to CREEP

This section serves the purpose to give an overview of the CREEP framework with some background on non-VLSI issues, such as microarchitecture and architecture simulator, while details on, e.g., implementation and power estimation will be given in the next section (Sec. III).

The CREEP framework contains two different components: 1) RTL code for the pipeline and level-1 (L1) caches, and 2) a modified SimpleScalar version as architecture simulator. Controlled via an extensive script, these components are configured and simulated with the goal to combine accurate per cycle resource statistics from the architecture simulator with power estimates resulting from an implementation of the RTL code.

The conceptual workflow of the framework is shown in Fig. 1 where two branches are visible. The most straightforward way to employ CREEP is to use the rightmost *simulation branch*. Here, the user controls parameters primarily associated with the caches via the script in CREEP:START and relies on energy values that we have already prepared. Since this use scenario requires no licenses for neither benchmarks nor design software, all users can use this branch.

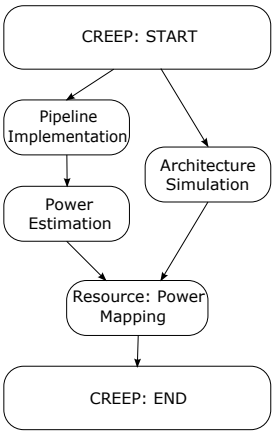


Fig. 1. The methodology embodied in the CREEP framework.

All energy values that CREEP is shipped with were prepared by us using the leftmost *implementation branch* which allows a user[1] to organize, generate, verify, and analyze both RTL code and netlists. Although this branch is not as approachable as the simulation branch, the CREEP framework was created in a modular manner to help implementation-centric users to, e.g., make revisions to portions of our current RTL code or remap our current RTL base to a different technology node to obtain a different set of power and energy estimates.

A. Pipeline Microarchitecture – 5SP

In this section, a microarchitecture compliant with MIPS I [12] called 5SP (the *5-Stage Pipeline*) will be presented. As the cornerstone of CREEP, 5SP is a simple microarchitecture, with limited pipelining (5 stages) and single-issue, in-order execution. While the CREEP framework is prepared with energy numbers annotated to all parts of the pipeline (making most users prefer the simulation branch), the purpose of using a simpler pipeline is to make it possible for a user to perform experiments also on the implementation branch.

Since the pipelining in 5SP is limited there are few opportunities to evenly distribute the pipeline logic between the different stages. In a perfectly balanced n-stage pipeline the cycle time of the design is roughly $1/n$ of the cycle time of a corresponding non-pipelined design [13]. But since data and instructions accesses need to retrieve data from SRAM banks, memory accesses are significantly slower than logic delays, which makes the critical path of some pipeline stages significantly longer than for others[2].

The implemented 5SP microarchitecture features some 50 instructions including different branches, logic and memory instructions and a register file with general-purpose 32-bit registers. 5SP does not include a floating-point unit to provide floating-point support, which was motivated by the targeted embedded domain where floating-point operations usually are replaced by fixed-point calculations.

An overview of the 5SP pipeline is shown in Fig. 2. The instructions are processed in five stages; instruction fetch (IF), instruction decode (ID), execute (EX), memory access (MEM), and write back (WB). In the IF stage, instructions are read from the level-1 instruction cache (L1IC) from an address pointed to by the PC register, which is updated to point to consecutive instructions or to branch target addresses. During the ID stage the register file is accessed and control signals for later stages are set based on the instruction type. Branch and jump instructions are solved in the ID stage, but by the time they are resolved the next instruction has already been fetched. A branch delay slot is utilized to solve this problem and is accounted for by the compiler. In the EX stage, arithmetic and logic operations are executed in an ALU. A dedicated two-stage multiplication unit is also available, spanning the EX and MEM stages. In the memory-access stage, loads and stores access the level-1 data cache (L1DC). Finally, in the WB stage, results are written back to the register file. In CREEP's RTL code, the MEM and WB stages are combined to simplify the implementation. However, the combined stage logically functions as two separate stages.

A hazard-detection unit, which physically resides in the ID stage but is for simplicity not shown in Fig. 2, detects any potential hazards and stops the pipeline by stalling the IF stage. In this manner, NOPs are inserted into the pipeline. The cache also produces a stall signal, which is asserted upon a cache miss. In contrast to the hazard stalls, cache misses stall the entire pipeline. The 5SP microarchitecture does not support exceptions, but these are by design rare events. Exceptions are necessary to support I/O, recover from errors (Invalid Opcode etc.), and for system calls.

The 5SP implementation includes level-1 caches, i.e., L1IC and L1DC, which are separate from each other (Harvard style) to avoid structural hazards. No L2 cache is included in the implementation. Instead an ideal memory module serves as

[1]Here, the workflow depends on the user's access to design software, design kits and cell libraries.

[2]Note though that a more substantial performance overhead is caused by stall cycles due to the dependencies between instructions (hazards) moving down the pipeline [14].

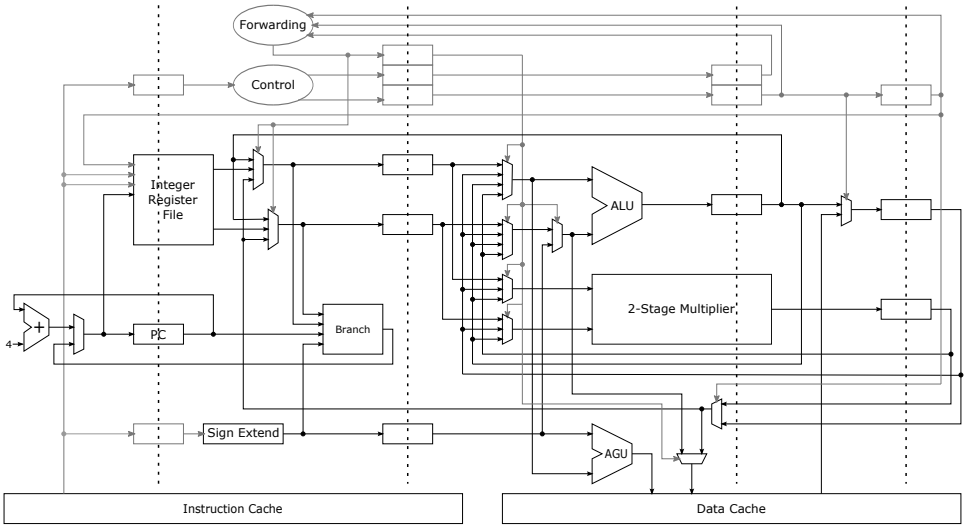


Fig. 2. Microarchitectural overview of the 5-stage pipeline (5SP) embedded in CREEP.

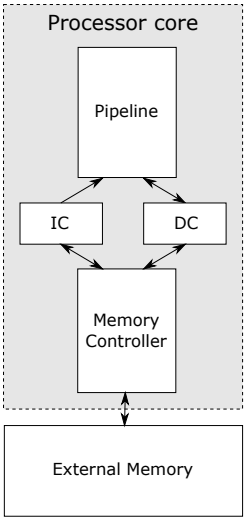


Fig. 3. Memory hierarchy of the 5SP.

a replacement for the lower levels of the memory hierarchy as shown in Fig. 3. The L1DC is available for read and write accesses, while the L1IC only serves reads. However, the L1IC still needs to access external memory on cache fills and in the case of a cache miss. The two caches share one memory bus to the external memory and a memory controller (arbiter) orchestrates which cache is allowed to access the external memory. The L1 cache RTL code is designed to be relatively flexible, as will be demonstrated in Sec. V.

B. Architecture Simulator – SimpleScalar

SimpleScalar is a wide-spread execution-driven architecture simulator [15] which was chosen because of the relatively simple processor pipeline that it models, it it easy to access and easy to use, and it is suitable for hardware-oriented studies thanks to its hardware-centric architectural model. The fact that SimpleScalar is execution driven is essential, as this makes it possible to capture the dynamic behavior of the underlying architecture in terms of branches and cache misses, which can have a dramatic impact on performance and energy. Furthermore, because SimpleScalar captures both the functionality and the performance of the architecture, validation is possible and accurate resource usage and time measurements (execution time in clock cycles) are made possible.

SimpleScalar supports configuration through configuration files that are provided when calling it from the command line [15]. These files enable features such as branch resolution, cache parameters, speculative execution, decode width, issue width and number of functional units to be tweaked without the need to rebuild the simulator.

In fact, SimpleScalar provides several different simulators of varying detail and speed [15]. The simplest and fastest simulator, called sim-fast, is a purely functional simulator which is not cycle-accurate. In contrast, the most complex simulator, the sim-outorder, supports out-of-order, speculative execution, and multi-issue while being cycle-accurate. The simulator used in the CREEP framework is based on a modified version of the sim-outorder simulator. The modifications were implemented to reduce the out-of-order pipeline modeled to an in-order pipeline that matches the 5SP pipeline microarchitecture described in Sec. II-A. The base simulator was then augmented with performance counters that tracked usage of pipeline resources relevant to energy estimation.

III. CREEP Components and Integration

While the previous section gave an overview of the CREEP framework and how it is used, this section will present more details on CREEP components and their interaction. Each subsection below corresponds directly to one node in the workflow of Fig. 1.

A. CREEP:Start – Configurability

The CREEP framework supports configurability that needs to be implemented identically in the RTL code and in the architecture simulator, since the simulator captures the behavior of the RTL and vice versa.

Since the energy per event is higher and since the events are more infrequent, cache parameters like size and associativity are the parameters that have the biggest impact on overall energy. Thus, the user configurability built into the framework was concentrated on parameters for L1IC and L1DC. The RTL pipeline does not include a L2 cache, hence, the framework does not estimate the power dissipation of this cache. However, SimpleScalar can still be used to simulate the L2 cache's impact on performance and, indirectly, on energy estimates. Thus, settings related to the L2 cache were also included amongst the configurable settings. The supported settings are presented below in Table I.

B. Architecture Simulation

Since the sim-outorder simulator of SimpleScalar allows for accurate resource tracking, this is the starting point for the simulator used in CREEP. Modifications to SimpleScalar's source code were performed with the goal of reducing the

TABLE I
CREEP SETTINGS

Unit	Setting	Values
L1DC	Associativity	1‡/2/4
L1DC	Replacement policy	LRU/Pseudo random
L1DC	Number of sets	64/128/256/512
L1DC	Line size (words)	8
L1DC	Bank size (words)	4/8
L1IC	Associativity	1‡/2/4
L1IC	Replacement policy	LRU/Pseudo random
L1IC	Number of sets	64/128/256/512
L1IC	Line size (instructions)	8
L1IC	Bank size (instructions)	4/8
L2DC	Latency	10/12/14
L2IC	Latency	10/12/14

‡: An associativity of 1 is equivalent to a direct-mapped cache.

modeled pipeline's complexity to a state that matches the pipeline's RTL code. The modifications included moving store operations from the commit stage to the issue stage (similar to loads) whereby the stores were locked to non-speculative in-order execution. Other features such as multi-issue, speculative execution (for instructions besides stores) and branch prediction were disabled through configuration files. However, an exception was made for the issue width which was set to two. This was necessary as load and store instructions are split up into a separate address calculation and a read/write instruction. This will not cause any issues for other instructions, as only one instruction at a time leaves the preceding dispatch stage. The branch prediction was configured to perfect, which resembles the branch delay slot technique used in the RTL code of the 5SP pipeline.

A common ground between the architecture simulator and the RTL code was found in the caches. The SimpleScalar caches are configured through the aforementioned configuration file and support a superset of the settings available in the RTL code. The SimpleScalar caches were configured to mirror the caches in the RTL code. All relevant settings for the framework are summarized in Table II.

TABLE II
RELEVANT SIMPLESCALAR SETTINGS

Setting	Description	Value
-fetch:ifqsize	Instruction fetch queue size, set to 1 to model a single-issue pipeline.	1
-decode:width	Decode width, set to 1 to model single-issue pipeline.	1
-issue:width	Issue width, set to 2 to model single-issue and allow loads/stores to be issued in one cycle.	2
-commit:width	Commit width, set to 1 to model a single-issue processor.	1
-issue:inorder	Pipeline is set to issue in-order.	true
-issue:wrongpath	Pipeline is set to issue non-speculatively.	false
-bpred	Branch predictor component set to perfect to emulate delayed branch slot.	perfect
-cache:il1 / -cache:dl1	L1 cache settings, to be coordinated with RTL.	-RTL
-cache:il1lat / -cache:dl1lat	L1 cache access latency	1
-cache:il2lat /-cache:dl2lat	L2 cache access latency	12

One major difference between the provided 5SP RTL code and the corresponding pipeline model implemented in SimpleScalar is that the latter supports a larger set of instructions, e.g., floating-point instructions and system calls. The impact of this difference was limited by choosing simulator benchmarks that included few of the unsupported instructions and features.

Several workloads were considered for the framework. MiBench is open source and consists of a set of 35 embedded applications which are divided into six suites each targeting a specific area in the embedded domain [16]. MiBench was chosen as the default workload for the framework, mainly because it is open source and can be shipped with the framework. However, not all 35 benchmarks in the suite are used because of incompatibility with the simulator. Instead, a representative subset of 20 benchmarks were selected from the automotive, consumer, network, office and security categories. The categories and pertaining benchmarks are shown in Table III.

TABLE III
MIBENCH BENCHMARKS

Category	**Applications**
Automotive	basicmath, bitcount, qsort, susan
Consumer	jpeg, lame, tiff
Network	dijkstra, patricia
Office	ispell, rsynth, stringsearch
Security	blowfish, rijndael, sha, pgp
Telecomm	adpcm, crc32, fft, gsm

C. Pipeline Implementation

The implementation branch of Fig. 1 is based on an RTL implementation of the 5SP microarchitecture (Sec. II-A), which provides the user a baseline processor on which it is possible to explore RTL variations. We have, however, already completed one 5SP implementation down to and including place-and-route, to generate 65-nm energy data for users that prefer only to work with the simulation branch of CREEP. This subsection will describe the implementation flow used to obtain these energy data and should be instructive for any user who wishes to perform a similar analysis.

The 5SP RTL code has gradually been developed during several recent projects [17], [18]. Since the integration effort for a processor pipeline is substantial, a solid verification methodology is imperative. To verify the RTL design, the Cadence Incisive Enterprise Simulator (IES) system [19] was used for logic simulation of executables, compiled for the MIPS I ISA, running in a testbench in which the 5SP RTL code was instantiated. Logic simulations of large designs are time consuming, but can be facilitated through the use of small and effective workloads with good test coverage. Since the EEMBC benchmark suite [20] targets embedded processors, is lightweight, and utilizes fixed-point arithmetics, the following benchmarks are used in the implementation branch of CREEP[3]: Autocorrelation, Convolutional Encoder,

[3]Any user who wishes to use scripts for running EEMBC needs to obtain the proper licenses from the EEMBC consortium [20].

FFT/IFFT, Viterbi Decoder, and RGBCMY01 (Consumer RGB to CMYK).

Timing-driven synthesis was performed using Synopsys Design Compiler (DC) [21] for a 65-nm LP low-V_T cell library, using worst-case corners, 1.1 V, and 125°C, and with a corresponding library for the SRAM memories in the caches. We used automatic clock gating to reduce the design's dynamic power dissipation. The synthesis was carried out for increasingly stricter timing constraints to find the maximum achievable clock rate and the worst-case design was established to meet a timing constraint of 2.5 ns, producing a netlist running at 400 MHz. The netlist verification was done using the same testbench developed for RTL verification above.

Place-and-route (P&R) using Cadence Encounter [22] was used to produce a post-layout netlist. While a post-synthesis netlist may suffice in some design-exploration situations, P&R was deemed necessary to be include not primarily because of the higher accuracy this stage provides but because the utilized SRAM memories are already placed and routed and, thus, are taking into account the routing's impact on power dissipation. However, since the extensive use of heuristics makes the P&R solution of each pipeline design differ significantly, here we can identify a conflict with the desired scalability of the flow. In CREEP, therefore, a more scalable approach was implemented that estimate the P&R impact on power dissipation by comparing the power of one placed and routed netlist to a post-synthesis netlist; from this comparison, a scaling factor could be deduced. The rationale behind this approach was that the pipeline logic was not subjected to any modifications but that it remains relatively unaffected by the configuration of the caches.

D. Power Estimation

Similar in purpose to Sec. III-C, this section describes the flow used to perform power estimation and it is included to help other users who wish to perform a similar analysis.

Based on the netlist obtained after implementation, the power consumption was estimated by using use-case simulations, in which switching activities for the nodes in the design were obtained. During SAIF generation the average switching activities of all netlist nodes are recorded throughout simulation, which allows an average power estimate of the design to be produced. (While VCD-based methods allow for detailed analysis of the power dissipation, the usage of VCD is computationally complex and hence less scalable than the SAIF-based method.) Cadence IES was used to simulate the netlists using the testbench and the EEMBC benchmarks from Sec. III-C as stimuli.

Synopsys PrimeTime PX (PT) [23] was used to generate the final power estimates. Synopsys PT was first used to remap the gate netlist to a different cell library from the library with worst-case conditions used during timing-driven synthesis. The power dissipation was analyzed for a library with nominal process corners, nominal VDD (1.2 V) and nominal temperature (25°C). This approach yields nominal power estimates for the pipeline design and, thus, allows different pipeline configurations to be compared under normal circumstances where the technology parameters are at their expected values.

The power estimation was done by reading the netlist and each of the aforementioned SAIF files. Thus, a total of five power reports, one per benchmark, were produced and averaged to create the final design power estimate. Hierarchal reports were produced for the design and the granularity of these reports was tweaked to reveal major pipeline units and to suit the performance counters generated by the simulator component (Sec. II-B).

Average power reports amortize the power of certain units over the power estimation time interval. Units such as the ALU, multiplier and L1DC are associated with enable signals that prompt them to activate, i.e., start switching and dissipating power. Unless the power of these units are scaled according to their usage, the framework would greatly underestimate their contribution to the final energy results. The solution to this problem requires information of how many cycles each unit in question is active during the estimation interval. This usage information was obtained by augmenting the RTL testbench used during verification and power estimation with counters that were incremented when the enable signal for a unit was asserted. Each unit counter was then divided by the total number of cycles also tracked by the testbench, to obtain the utilization factor U as

$$U = \frac{\#active\_cycles}{\#total\_cycles} \tag{1}$$

The power dissipation reported by Synopsys PT ($P_{unscaled}$) was then divided by U, as shown in Eq. 2 to obtain the final power values used in the framework.

$$P_{scaled} = \frac{P_{unscaled}}{U} \tag{2}$$

Since the SRAM memories come placed and routed in macros, scaling to consider the power impact of place and route (P&R) must be applied to all other pipeline units. As discussed in Sec. III-C, the challenge is to use P&R data in a scalable manner. Thus, in an attempt to streamline the P&R scaling, one archetype combinatorial pipeline unit was selected and the power of the unit was compared post-synthesis and post P&R. The ALU was chosen and the power estimates were obtained by assigning switching activities on the ALU input and let these propagate through the unit. The power was then extracted using Synopsys PT and a factor between the post-synthesis and post-layout was obtained as shown in Eq. 3. This factor was then used to scale the post-synthesis power estimates for all pipeline units.

$$P\&R_{scaling} = \frac{P_{ALUpost-synthesis}}{P_{ALUpost-layout}} \tag{3}$$

A similar estimation was done for the clock-tree power, which is very limited in a post-synthesis netlist where bare register clock pins are exposed directly to the simulator. The clock power dissipation in a post-layout netlist is considerably larger since a clock tree has been inserted to drive all register clock pins.

TABLE IV
SIMPLESCALAR PERFORMANCE COUNTERS

Stage	Unit	Condition
IF	Pipeline logic	No stall
IF	L1IC	No stall
DE	Pipeline logic	No stall
DE	Decode logic	No stall
DE	Register file	Instruction has operands
DE	Hazard detection	No stall
DE	Branch logic	Control instruction
EX	Pipeline logic	No stall
EX	ALU	ALU instruction
EX	Multiplier	Mult instruction
EX	AGU	Load/store
MEM	Pipeline logic	No stall
MEM	L1DC	Load/store
MEM	LSU	Load/store
WB	Pipeline logic	No stall, instruction produces result

E. CREEP:END – Combining Circuits and Architecture

One challenge with combining the two CREEP components to form one complete workflow is to accurately match measurement units and pipeline building blocks between the two abstraction levels.

Synopsys PT reports power dissipation in the SI unit for power, i.e., Joules per second rather than Joules per cycle which is required for the power estimates to be combined with the resource counters reported by SimpleScalar. The conversion necessary is relatively straightforward to make, since the clock rate is assumed to match the timing constraint set during synthesis: $E_c = P/f$ where the power dissipation P is in Joules per second, E_c is in Joules per cycle, and f is the clock rate.

The most complex issue when combining the pipeline implementation with SimpleScalar is to appropriately match the building blocks in the pipeline with pipeline constructs that exist in the simulator. There is indeed a practical limitation as to how fine grained this mapping of pipeline resources can be done given that SimpleScalar uses a relatively high level of abstraction, mainly exposing discrete pipeline stages and major architectural events coupled to each of these. The power reports produced by the CREEP's RTL branch were adapted to match the more coarse-grain resources modeled by the simulator. As the demonstration will later show (Sec. V), the resources modeled this way are efficiently representing the pipeline's most energy-dissipating blocks.

Performance counters for each pipeline stage were introduced in the simulator to reflect the pipeline logic. Selective performance counters that appreciate that instructions do not use all resources in each pipeline stage were also introduced for pipeline units. One additional performance counter was added in relation to the L1IC to emulate way-determination [24], which is an optimization technique that is otherwise not present in the 5SP RTL implementation. The tracked units are shown in Table IV.

IV. FRAMEWORK AUTOMATION

The framework RTL and simulator components create an intimidating amount of output data that need to be combined as discussed in Sec. III-E. By automating the framework and presenting the user with a set of configurable parameters, the framework becomes user friendly and, more importantly, the pipeline energy evaluations become reproducible.

The automation is handled via a script that, when invoked, runs the entire workflow. The scripting language chosen to implement the script was Perl which supports text handling and file I/O. To ensure that the architectural parameters are the same for the RTL and simulator, all parameters (save for the configurable settings) were hidden from the user. In addition to the configurations listed in Sec. III-A, a setting used to enable or disable the emulated way-prediction in the L1IC was added. Configuration files, called CREEP configurations, were created to list the configurable options. As users would want to create several configuration files, the main script was designed to use one such configuration as argument when calling the script as shown below:

```
./CREEP.pl -[CREEP_configuration]
```

The parameters specified in the configuration file are here parsed by the main Perl script, applied to the RTL code, the simulator configuration and all auxiliary scripts surrounding these components such as RTL testbenches, Synopsys DC synthesis script, SAIF-generation scripts and Synopsys PT scripts. The main script then starts both components and makes use of the outputs, i.e., power estimates from the RTL and resource usage information from the simulator. The implementation output would then be scaled by the main script and combined with the resource mapping discussed in Sec. III-E to produce the final energy estimates.

The CREEP script was designed to offer the user three different options; 1) run the entire framework, i.e., both branches, with both RTL and simulator components, 2) run the implementation branch alone and 3) run the simulation branch separately. The options were designed to be specified when invoking the script from a terminal as such:

```
./CREEP.pl -[flag] -[CREEP_configuration]
```

Allowing separate components to be used allows users that because of licensing issues are unable to use the RTL component to still use the framework. An issue here is that simulator statistics alone are not enough to produce the final energy results. This issue was addressed by saving the partial results generated from the RTL and simulation components for later use. The main script would then be able to load these partial results when running one of the components. It was decided to name the partial results after the provided CREEP configuration file to allow the aforementioned command invocation to remain unaltered and the interaction with the main script simple.

The complete workflow is shown in Fig. 4. The workflow starts with the user supplying an option flag and a CREEP-configuration file as shown in Sec. IV. The main script, named `CREEP.pl`, applies the configuration to all relevant files in the

framework and starts the components designated by the flag. The RTL and simulator components generate data, i.e., power scaling information, pipeline power dissipation estimates and resource counters, which are combined by the `CREEP.pl` script according to the power scaling and resource mapping discussed in Sec. III-E.

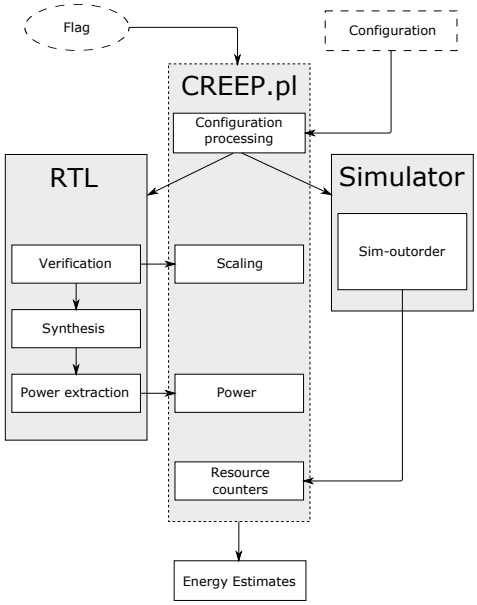


Fig. 4. The workflow of the framework showing the RTL and simulator components and the central CREEP.pl script.

Due to licensing issues, the RTL component is shipped without EDA tool binaries and cell libraries, i.e., no logic nor SRAM libraries are included in the framework package. However, as discussed in Sec. IV, a selection of configurations and pertaining power estimates are supplied with the framework to allow the simulator component to function separately and produce energy estimates of the provided configurations.

V. Demonstration of Framework

We use a selection of CREEP configurations to demonstrate the framework. Due to the large design space supported, the number of configurations had to be limited to those shown in Table V. These were selected on the basis that they span the design space supported by the framework, i.e., cache sizes and associativity. Unless specified, all configurations use a 12-cycle L2 cache access latency (L2L). Furthermore, all caches use a line and bank size of 8.

8kB 1-1 represents a lightweight embedded processor, while *8kB 2-2* and *8kB 4-4* are chosen to evaluate the impact of increased associativity on performance and energy. Because instructions has more regular access patterns than data, an L1IC often has lower associativity than an L1DC has. To evaluate lower associativity in the L1IC, which will reduce power dissipation while sacrificing some performance, the *8kB 1-4* configuration is chosen. Variations in L2 access latency, which impact execution time and thus energy, are evaluated via *8kB 1-1 10*, *8kB 1-1 14*, *8kB 2-2 10*, *8kB 2-2 14*, *8kB 4-4 10*, and *8kB 4-4 14*. Again it should, however, be noted that

TABLE V
Cache Parameters for Selected Configurations

Name	Size	Assoc.		RP	#Sets	#Sets	L2L
		IC	DC		IC	DC	
8kB 1-1 10	8kB	1	1	N/A	256	256	10
8kB 1-1	8kB	1	1	N/A	256	256	12
8kB 1-1 14	8kB	1	1	N/A	256	256	14
8kB 2-2 10	8kB	2	2	LRU	128	128	10
8kB 2-2	8kB	2	2	LRU	128	128	12
8kB 2-2 14	8kB	2	2	LRU	128	128	14
8kB 4-4 10	8kB	4	4	LRU	64	64	10
8kB 4-4	8kB	4	4	LRU	64	64	12
8kB 4-4 14	8kB	4	4	LRU	64	64	14
8kB 1-4	8kB	1	4	LRU	256	64	12
16kB 4-4	16kB	4	4	LRU	128	128	12
16kB 2-4	16kB	2	4	LRU	256	128	12
16kB 2-2	16kB	2	2	LRU	256	256	12
32kB	32kB	4	4	LRU	256	256	12

CREEP does not include any direct energy estimates of the L2 cache. The 16kB configurations represent a performance-oriented embedded processor: *16kB 2-2*, *16kB 2-4*, and *16kB 4-4*. Likewise, the single *32kB* configuration represents a no-compromise performance-oriented embedded processor and uses the upper bound on the supported cache size in the framework.

A. MiBench Execution Times

Fig. 5 shows the execution time ($\#sim\_cycles \times T_{cycle}$) for the MiBench benchmarks that are primarily used. A few benchmarks deviate significantly from the average: As the name implies, `basicmath` is mainly composed of arithmetic operations and is by far the largest benchmark with 6,360,380,890 simulated instructions. In contrast, `rsynth` and `stringsearch` are included in the office category (Table III) and are composed of text processing computations. At 1,138,490 and 4,656,782 simulated instructions, respectively, these benchmarks are relatively small.

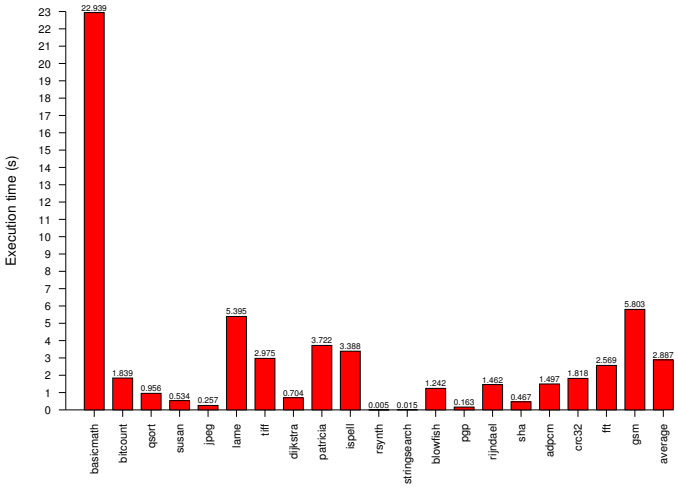


Fig. 5. Per-benchmark execution time for 16kB 4-4 configuration.

The number of simulated cycles is greater than the number of simulated instructions. This is because the simulator

component captures the pipeline's dynamic events, such as cache misses and hazards stalls, that degrade the processor's instructions per cycle (IPC). For `basicmath` the 16kB 4-4 configuration manages an IPC of 0.7 which offers further insight into the execution time of the benchmark.

B. Execution Time vs Power Dissipation

The average execution time versus the average power dissipation for all aforementioned configurations is shown in the scatter plot in Fig. 6.

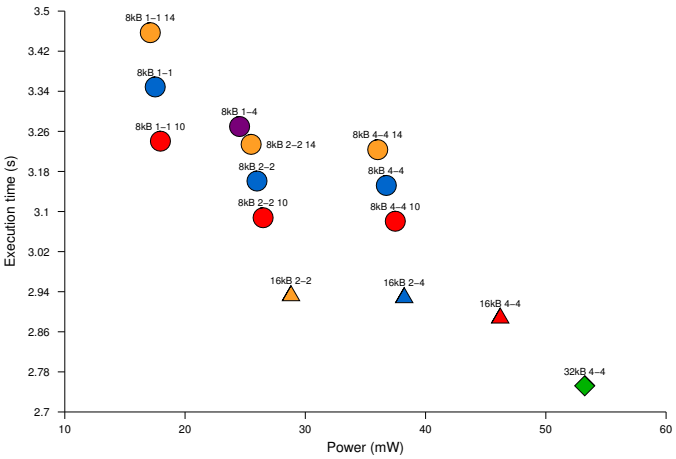


Fig. 6. Execution time versus power (MiBench).

As expected, the 8kB configurations have the lowest performance but also the lowest power dissipation. The small cache capacity causes relatively high cache miss rates, as demonstrated in Fig. 7. Each miss is associated with cycle penalties of accessing the L2 caches; L2L in Table V.

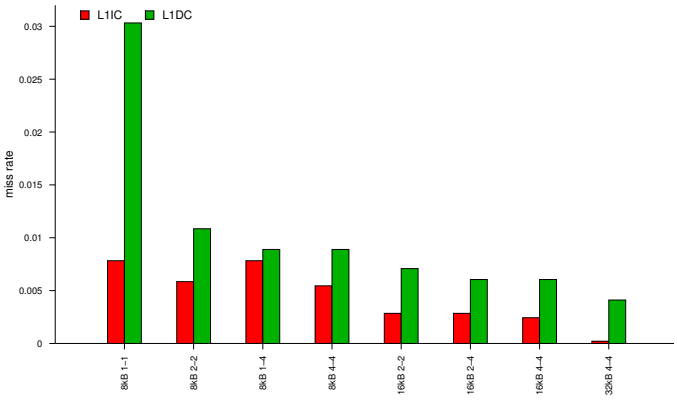


Fig. 7. Miss rates (MiBench).

The impact of L2L on performance is clearly visible in Fig. 6. The three L2L configurations (three for each of 8kB 1-1, 8kB 2-2, and 8kB 4-4) are evenly spaced on the performance axis; that with an L2L of 10 is the fastest, while that with an L2L of 14 is the slowest. The difference in power dissipation manifests because the L2 latency impacts how frequently the caches are accessed. The average cache power drops as L2 latency increases, since there is a higher proportion of stall cycles which dissipate less energy than regular cycles.

In order to enhance performance by reducing miss rates (Fig. 7), the L1DC associativity can be increased to four ways. As Fig. 6 shows this 8kB 1-4 configuration performs better than 8kB 1-1 (assuming equal L2L of 12 cycles), but it does so with a power dissipation penalty that is substantial.

Similarly, we can explore using the same associativity in both the L1IC and the L1DC, i.e., the 8kB 2-2 and the 8kB 4-4 configurations. In comparison to 8kB 1-4, 8kB 2-2 trades increased miss rates in the L1DC for decreased miss rates in the L1IC. Since the L1IC is accessed almost every cycle, lower L1IC miss rate has a larger impact on performance than lower L1DC miss rate, which explains why this configuration performs better.

The 8kB 4-4 configurations do not yield a very significant performance increase over 8kB 2-2, but they draw considerably more power since the hardware complexity increases with the increased level of associativity. For example, compared to the direct-mapped 8kB 1-1 configuration, the 8kB 1-4, 8kB 2-2, and 8kB 4-4 configurations require an additional six, four and 12 SRAM memories, respectively.

When stepping up the cache capacity to 16kB the performance increases substantially. Larger capacity caches can store larger parts of the program and more data without the need to evict items to the L2 caches. In effect, the number of misses and subsequent L2 accesses are reduced, which is shown in Fig. 7 where the 16kB configurations have lower miss rates in both caches compared to the 8kB configurations. The lowest performing 16kB configuration is the 16kB 2-2 configuration with the lowest degree of associativity. However, as shown in Fig. 6 the configuration offers good performance with comparatively low power dissipation that is substantially better than the other 16kB configurations. Moreover, the 16kB 2-2 configuration is both faster and has a lower power dissipation than the smaller 8kB 4-4 configurations.

Increasing the associativity in the L1DC to four-way yields the 16kB 2-4 configuration. The increase in associativity does increase performance but only slightly which is explained by the minor reduction in the L1DC miss rate shown in Fig. 7. In contrast, the power dissipation increases substantially because additional SRAM memories are added to facilitate the associativity, which shifts the configuration to the right in Fig. 6. Increasing the associativity in the L1IC results in the 16kB 4-4 configurations which manages to achieve the highest performance of all the 16kB configurations but also dissipates the most power.

As the capacity of the cache is increased to 32kB the performance rises considerably but less than the step from 8kB to 16kB. The increase in performance stems from the lower miss rates in the L1DC and especially the L1IC shown in Fig. 7. The 32kB cache clocks in at 2.75 s which is by a sizable margin the fastest configuration. However with eight large SRAM macros and low execution time the configuration dissipates the most power at around 53 mW.

The general trend that can be observed in the results indicates that higher capacity rather than increased associativity produces the most power-efficient configurations that also boasts good performance increases. This can be explained by the non-linear power increase of SRAM blocks, which

are designed to be dense and power-efficient. There are fixed costs associated with the input pins on the blocks and the internal power dissipation scales well with size. Thus using fewer larger SRAM blocks, i.e., lower associativity, is more power-efficient than several smaller SRAM blocks. Another trend is the diminishing returns in performance for increasing cache size and especially associativity, which could indicate that MiBench benchmarks are too simple to capture the performance increases normally associated with high capacity and associative caches.

C. Pipeline Energy Breakdown

The absolute energy integrated over all MiBench benchmarks for different pipeline configurations is shown in Fig. 8. The energy is divided into four categories; 1) clock network, 2) pipeline, 3) L1IC and 4) L1DC. Since the energy metric combines execution time and power dissipation, power-efficient designs are not necessarily the most energy-efficient if their performance is low and leakage is prominent. However, in this case the 8kB direct-mapped configurations turn out to be the most energy-efficient configurations (but whether it offers the desired performance is a different question).

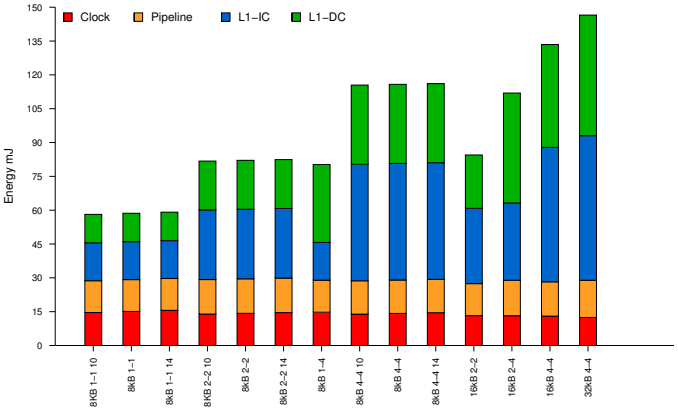


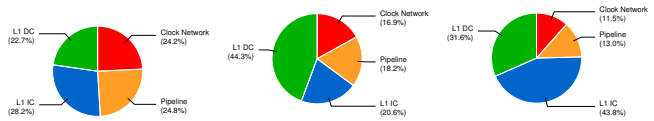
Fig. 8. Absolute energy of the different configurations.

A minor energy difference can be observed between the 8kB 1-1 configurations which is caused by the difference in L2 latency. Higher L2 latency causes longer execution times during which the clock network dissipates power. A similar reasoning on the L2 latency applies to the 8kB 2-2 configurations.

The 16kB 2-2 configuration manages to be the most energy-efficient 16kB configuration while also beating the 8kB 4-4 configurations and breaking even with the 8kB 2-2 configurations. It performs good enough compared to the 8kB 2-2 to compensate for its higher power dissipation and it beats the 8kB 4-4 configurations in both performance and power. The 16kB 2-4 configuration dissipated more power in the L1DC than the 16kB 2-2 configuration while only slightly increasing the performance and is, thus, less energy-efficient. The least energy-efficient caches are the large capacity and four-way associative 16kB 4-4 and 32kB configurations. The lower execution time offered by both configurations is insufficient to offset the higher power dissipation in the caches.

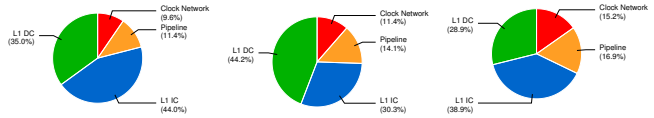
The general trends are that the caches dominate the energy consumption and the number of SRAM macros, i.e., associativity, is the main source to this. Cache size does contribute but to a lesser extent. The L1IC tends to consume the most energy when the caches are balanced (same associativity in the caches), which is expected as an instruction is ideally fetched each cycle. In contrast, the L1DC is accessed roughly every fourth instruction. The clock energy depends on the execution time and decreases with increasing performance (# cycles). However, the clock energy differences are small and really only observable between the 8kB 1-1 and the 32kB configurations. The energy of the pipelines remains relatively constant across the configurations; variations are most likely due to synthesis heuristics.

Assuming only the configurations with the default L2 latency of 12, we also show the energy distribution as pie charts in Figs 9-11. What is shown is that increasing cache sizes, with the exception of lower associative caches, shifts the energy distribution towards the caches. For the 8kB 1-1, 8kB 2-2, 16kB 2-4 and 16kB 2-2 configurations the energy instead shifts towards the clock network and the pipeline, most notably so for the 8kB 1-1 configuration as the cache energy has been reduced significantly when compared to the other ones.



(a) 8kB 1-1 (b) 8kB 1-4 (c) 8kB 4-4

Fig. 9. Energy distribution of three different 8kB configurations.



(a) 16kB 4-4 (b) 16kB 2-4 (c) 16kB 2-2

Fig. 10. Energy distribution of three different 16kB configurations.

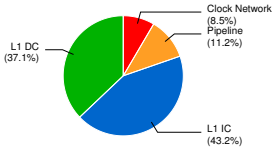


Fig. 11. Energy distribution of 32kB configuration.

VI. Related Work

Energy estimation/evaluation frameworks have over time evolved from estimation methods limited to specific structures within a processor, to complex system-level tools more or less detached from implementation details. While analytical estimation generally has the advantage of being scalable across different microarchitectures, empirical methods are based on detailed implementation information and thus are best suited for the type of architectures from which they were derived [5].

CACTI, released in 1996, specifically targets cache structures [6] and uses analytical models to estimate both power and delay within the cache structure. Since caches are highly regular structures, analytical models do not have to be that complex to accurately estimate energy and delay. In contrast to CACTI, CREEP also models the less regular datapath with which the caches are integrated.

Wattch and SimplePower, both released in 2000, analytically model power for a complete processor. Wattch links SimpleScalar to analytical power models [7] and bases its power estimations on a collection of parameterized power models for different hardware structures and per-cycle resource usage counts generated through cycle-level simulations using SimpleScalar. SimplePower is an execution-driven, cycle-accurate RTL energy estimation tool that uses a combination of analytical and transition-sensitive energy models for a five-stage pipeline [8]. However, developing transition-sensitive models is not straightforward and, thus, the pipeline control path was omitted because it was considered too challenging to model [25]. Compared to CREEP, Wattch and SimplePower, while being more flexible, fail to capture the implementation integration aspect that CREEP addresses.

The McPAT framework was released in 2009 and estimates power, area and timing [9]. In contrast to SimplePower and Wattch, McPAT is compatible with any performance simulator through an XML interface. Furthermore, McPAT is built on more accurate analytical models compared to Wattch and these models also include static power. Similar to McPAT, CREEP provides a system perspective but does so more accurately as power estimates are obtained from an RTL implementation. However, while McPAT supports multicore contexts, CREEP targets simple embedded processors.

In contrast to the above approaches, IBM's PowerTimer is based on empirical data collected from existing microprocessors [10]. Low-level circuit macros are analyzed and used to generate higher-level energy models for microarchitecture units [26] which can be controlled by technology and circuit parameters as well as microarchitectural parameters. CREEP is likewise limited to the specific architecture implemented in RTL. Both frameworks work at the system level but PowerTimer chooses to distance itself from the physical implementation through parameterized models which lends it greater flexibility at the expense of accuracy.

VII. Conclusion

To enable higher-level estimations, it is necessary to abstract away details pertaining to the technology and circuit level. The risk of abstraction is, however, that important dependencies at the circuit level are neglected, rendering the result of abstraction inaccurate as far as implementation is concerned. We have introduced CREEP which stands for *Chalmers RTL-based Energy Evaluation of Pipelines* [11]. The main goal of the framework is to deliver energy evaluation results that are faithful to the VLSI integration of the pipeline and its level-1 caches, while simultaneously keeping with the rational workflow that involves running a pipeline model in an architecture simulator with processor software benchmarks.

References

[1] M. Själander, M. Martonosi, and S. Kaxiras, *Power-efficient computer architectures: Recent advances*. Synthesis Lectures on Computer Architecture, Dec. 2014.

[2] T. Kuroda, "CMOS design challenges to power wall," in *Proc. of International Microprocesses and Nanotechnology Conference*, Oct. 2001, pp. 6–7.

[3] B. Davari, R. Dennard, and G. Shahidi, "CMOS scaling for high performance and low power - The next ten years," *Proceedings of the IEEE*, vol. 83, no. 4, pp. 595–606, Apr. 1995.

[4] P. Somavat and V. Namboodiri, "Energy consumption of personal computing including portable communication devices," *Journal of Green Engineering*, vol. 1, no. 4, pp. 447–475, 2011.

[5] S. Kaxiras and M. Martonosi, *Computer architecture techniques for power-efficiency*. Morgan & Claypool Publishers, 2008.

[6] S. Wilton and N. Jouppi, "CACTI: an enhanced cache access and cycle time model," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 5, pp. 677–688, May 1996.

[7] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," *SIGARCH Comput. Archit. News*, vol. 28, no. 2, pp. 83–94, May 2000.

[8] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "The design and use of SimplePower: a cycle-accurate energy estimation tool," in *Proc. of the 37th Annual Design Automation Conference (DAC)*, 2000, pp. 340–345.

[9] S. Li, J. H. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proc. of 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2009, pp. 469–480.

[10] D. Brooks, P. Bose, V. Srinivasan, M. K. Gschwind, P. G. Emma, and M. G. Rosenfield, "New methodology for early-stage, microarchitecture-level power-performance analysis of microprocessors," *IBM Journal of Research and Development*, vol. 47, no. 5.6, pp. 653–670, Sept. 2003.

[11] Chalmers RTL-based Energy Evaluation of Pipelines (CREEP). [Online]. Available: http://www.cse.chalmers.se/research/group/vlsi/CREEP/

[12] J. Hennessy, N. Jouppi, S. Przybylski, C. Rowen, T. Gross, F. Baskett, and J. Gill, "MIPS: A microprocessor architecture," in *ACM SIGMICRO Newsletter*, vol. 13, no. 4. IEEE Press, 1982, pp. 17–22.

[13] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2012.

[14] D. M. Harris and S. L. Harris, *Digital design and computer architecture*. Elsevier, 2013.

[15] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An infrastructure for computer system modeling," *Computer*, vol. 35, no. 2, pp. 59–67, 2002.

[16] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *Proc. of IEEE International Workshop on Workload Characterization*. IEEE, 2001, pp. 3–14.

[17] V. Saljooghi, A. Bardizbanyan, M. Själander, and P. Larsson-Edefors, "Configurable RTL model for level-1 caches," in *Proc. of IEEE NORCHIP*, Nov. 2012.

[18] M. Själander and P. Larsson-Edefors, "FlexCore: Implementing an exposed datapath processor," in *Proc. of International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIII)*, July 2013, pp. 306–313.

[19] *Incisive Enterprise Simulator (IES)*, Cadence Design Systems, Inc., July 2011.

[20] Embedded Microprocessor Benchmark Consortium. [Online]. Available: http://www.eembc.org

[21] *Design Compiler®*, Synopsys, Inc., Mar. 2010.

[22] *Encounter® Digital Implementation (EDI)*, Cadence Design Systems, Inc., July 2011.

[23] *PrimeTime® PX*, Synopsys, Inc., June 2011.

[24] B. Calder, D. Grunwald, and J. Emer, "Predictive sequential associative cache," in *Proc. of International Symposium on High-Performance Computer Architecture (HPCA)*, Feb. 1996, pp. 244–253.

[25] N. Vijaykrishnan, M. Kandemir, M. J. Irwin, H. S. Kim, and W. Ye, "Energy-driven integrated hardware-software optimizations using SimplePower," in *Proc. of the 27th Annual International Symposium on Computer Architecture*, 2000, pp. 95–106.

[26] D. Brooks, P. Bose, and M. Martonosi, "Power-performance simulation: design and validation strategies," *ACM SIGMETRICS Performance Evaluation Review*, vol. 31, no. 4, pp. 13–18, 2004.