High Level Power and Energy Exploration using ArchC

T. Gupta, C. Bertolini, O. Heron, N. Ventroux
CEA LIST
PC94, Gif-Sur-Yvette, F-91191, France
Email: tushar.gupta@cea.fr

T. Zimmer, F. Marc
Université Bordeaux I
351 cours de la Libération
33405 TALENCE cedex, Bordeaux, France
Email: thomas.zimmer@ims-bordeaux.fr

Abstract

With the increase in the design complexity of MPSoC architectures, estimating power consumption is very complex and time consuming at lower level of abstraction. We propose a methodology using ArchC named Power-ArchC for a fast high-level estimation of processor power consumption. Power values are obtained by an instruction level power characterization at gate level. The requirements for power evaluation infrastructure are compatible processor models written in ArchC and RTL, and the Technology library. We show power results for a 32-bit MIPS processor with different benchmarks, based on 45nm technology.

1. Introduction

The emergence of new embedded applications for telecom, automotive, digital television and multimedia applications has fueled demand for architectures with higher performances, more chip area and high power efficiency. These applications are usually computation-intensive, which prevents them from being executed by general-purpose processors. Thus, designers are showing interest in a System-on-Chip (SoC) paradigm composed of multiple processors and a network that is highly efficient in terms of latency and bandwidth. The resulting new trend in architectural design is the MultiProcessor SoC (MPSoC) **[1]**. MPSoCs' architectures can have homogeneous or heterogeneous processors, depending on the application requirements. Choosing the best processor among hundreds of available architectures, or even designing a new processor, requires the evaluation of many different features (pipeline structure, ISA (Instruction Set Architecture) description, register files, processor size...), and the architect needs to explore different solutions in order to find the best trade-off. The processor Instruction Set Simulator (ISS) has an important role, and must have the following features: it should be able to parameterize, fast and accurate, and able to be easily integrated in the MPSoC simulation environment.

The ISS emulates the behavior of a processor by executing the instructions of the target processor while running on a host computer. Depending on the abstraction level, it can be modeled at the functional or cycle-accurate level. On the one hand, the functional ISS model abstracts the internal hardware architecture of the processor (pipeline structure, register files...) and simulates only the ISA. Therefore, it can be available in the early phase of the MPSoC design for the application software development, where the simulation speed and the model development time are important factors for a fast design space exploration. On the other hand, the cycle-accurate ISS model simulates the processor at an abstraction level between the RTL and the functional model. It presents most of the architectural details that are necessary for processor sizing, in order to evaluate in advance its performance capabilities in the MPSoC design. All these advantages come at the expense of a slower simulation speed and longer development time.

As a corollary to Moore's law, power consumption of computing nodes doubles every 18 months **[14]** and needs to be evaluated as soon as possible in the design stages. Due to this, there are many tools at present to assess power at different levels of abstraction. Most commercial tools start working at the design after synthesis. Such tools are not candidates for MPSoC power analysis. The different power analysis methodologies and tools will be discussed in the Literature review, in Section 2. In this paper, we propose '*Power-ArchC*' that is a powerful ISS enabling the power analysis of processor cores at functional abstraction level. Compared to previous works, it is the first work that enables the generation of a power aware

ISS ready to be integrated in a complex SystemC based SoC design with a short development time. The technical contributions of this paper are:

- A semi-automatic design flow extended with power exploration capabilities at High-Level.
- A Power aware ISS generated by an Architecture description language.
- An experimental analysis of Energy and Power of a 32-bit MIPS processor **[30]**.

Section 2 will provide motivation behind our work and present some already existing methodologies at different levels of abstraction. Then we will present our methodology to estimate power at functional level in Section 3. We apply the proposed methodology to the MIPS case study and provide results in Section 4. Finally, a conclusion will be available in Section 5.

2. Related Work

Lot of research has been performed in power estimation techniques and tools at transistor level, gate level and register-transfer level **[1, 2, 3, and 4]**. Also in recent days, more research is performed at cycle-accurate and behavioral levels **[7]**. Since they target different levels of detail, they make different trade-offs between simulation time and accuracy. Some recent academic tools like *'Wattch'* **[7]**, *'SoftExplorer'* **[34]** and others enable the power analysis at front end design. Most simulators are parameterized, so they can be used to estimate the energy consumption of different system configurations. In the rest of this section, we will discuss about some already existing methodologies and tools (commercial or academic).

At *Transistor level*, power estimation is typically performed as a by-product of circuit simulation **[4]**. It enables the most accurate estimation but in detriment of a very slow speed because the simulation is performed late in the design cycle. These simulators characterize models of transistors and estimate voltage and current behaviors over time. Power dissipation of transistors comes from three sources: switching power, short-circuit power, and leakage power. Such simulation is time-consuming but useful in integrated circuit design. Transistor-level simulators such as *HSPICE, HSIM, ELDO, SPECTRE,* are not suitable in evaluating power consumption of large programs on complex systems, as they need large amount of memory space for storing all the details.

Gate-level approaches simulate a gate-level design, and calculate power by considering the switching activity (average number of toggling per time unit), the time, the equivalent capacitance and voltage of internal nodes **[5, 6]**. Compared to the previous approach, power values are less accurate but simulation speed increases. Techniques for power estimation and switching units of a circuit can be mapped to predefined minimized activities. At gate and circuit levels, it is usually classified as (1) statistical, or (2) probabilistic. In statistical methods, circuit simulation is performed using a set of randomly chosen input vectors, while monitoring the switching activity on each circuit node. The simulation runs until the switching activity converges to the average switching activity. Convergence is tested by a statistical mean estimation technique, such as the Monte Carlo procedure **[2, 6]**. In probabilistic methods, user-supplied input probabilities are propagated into the circuit to produce signal probabilities at every node. The switching activity may be determined once the signal probabilities at each node are computed. An example is '*PrimeTime'* **[31]**, which delivers a dynamic and leakage power analysis for design geometries at 90-nm and below. Designers have a single, unified analysis environment for timing, signal integrity and power analysis that is anchored by the *PrimeTime static timing* solution.

At *Cycle-accurate level*, power model considers the equivalent capacitance of a macro logic block. Compared to the previous approach, accuracy is lower because the internal switching activity of the block is not considered, only I/O toggling activity. Conversely, simulation speed is faster. At Cycle-accurate-level simulator simulates the execution at the level of individual cycles, allowing keeping track of power behavior changes across cycles. Examples of cycle-level simulator are *'Wattch'* **[7]**, *'SimplePower'* **[32]** and *'Sim-Panalyzer'* **[9]**. *Wattch* and others works at cycle-accurate level. *Wattch* involves analytical capacitance models which have to be developed for each block of a processor by the microarchitecture which is sometimes very difficult to perform. Indeed, at the cycle-accurate level, the processor's behavior is simulated cycle by cycle. In these tools, it is not a problem when only a small portion of the code (a few instructions) is simulated, but this may be very time consuming for large programs. Moreover, cycle-level simulations necessitate a low-level description of the architecture.

Instruction-level simulators provide coarser power behavior than the cycle-accurate ones. It is possible to analyze very quickly the power consumption of the circuit but with a rather low accuracy than descriptions made at lower abstraction level. The simulation is based on the instruction-level energy profiling of the instruction set of the target processors and the assumption that the energy consumption of an instruction is mostly independent of the addressing mode or operands or previous instructions. Some methodologies about Instruction level power analysis are discussed in recent papers **[10, 11].** One of the instruction-level simulators is '*Joule-Track'* **[33]**. *Joule-Track* is available as an online resource and has various estimation levels. It acts as a predictor for a set of benchmark programs evaluated on the StrongARM SA-1100 and Hitachi SH-4 microprocessors.

At *Application-level*, to estimate whole-program power-consumption, the methodologies generally work as predictor of power consumption for a program **[12]**. An application-level simulator, *SoftExplorer* **[34]** works at C-level and relies on the Functional-Level Power Analysis, which results in a power model of the processor. *SoftExplorer* is an interesting tool that works at C-level and relies on the Functional-Level Power Analysis, which results in a power model of the processor.

F. Klein et al. propose *PowerSC* **[35]**, a power-aware extension of SystemC classes. It allows power estimation of circuits described in SystemC language. *PowerSC* is based on a new multi-model power estimation engine. It selects the macromodeling technique leading to the least estimation error for a given system component depending on the properties of its input-vector stream. Design effort for power insertion in SystemC design is relatively limited: add of macros in user code and, technology and power libraries that are automatically built by *PowerSC* flow.

We provide this new methodology and tool proposed in this paper, called *Power-ArchC* because the tools at transistor, gate and cycle-accurate levels are not a practical solution to perform power explorations for a complete processor, whereas *Joule-Track* is not suitable for SystemC based processor designs written in *ArchC* language. In addition we need to estimate power of dynamic applications that will be executed on a processor; this is not possible with *SoftExplorer*. Although *PowerSC* is a very powerful framework that would fit to our needs, its availability is currently limited to gate-level description. *Power-ArchC* is based on a power-reuse model approach. Power values of instructions are obtained at gate level for better accuracy while simulation time for characterization remains acceptable. The instruction set architecture is annotated with obtained power values in the simulator.

3. High-Level power analysis using ArchC

Power-ArchC is built on the top of the *ArchC* methodology. *ArchC* provides an efficient framework for describing a processor architecture and ISA at behavioral or cycle accurate level. In addition, it automatically generates an ISS with a short development time, but it does not have any way to estimate its power consumption. We have extended the *ArchC* methodology with power capabilities. Using power model from Instruction Level Power Characterization (ILPC) into *Power-ArchC*, we can provide total energy used and average power consumed for a given benchmark, design implementation and technology node. *Power-ArchC* is built into four steps: *ArchC* based description, RTL design and synthesis, ILPC and back-annotation. We first present some preliminaries about *ArchC* and related ISS and then *Power-ArchC* methodology.

3.1. ArchC based ISS

An Instruction Set Simulator (ISS) **[36]** is a simulation model, usually coded in a high-level programming language, which mimics the behavior of a mainframe or microprocessor by "reading" instructions and maintaining internal variables which represent the processor's registers [36]. The ISSs enable fast design space exploration and simulation of complex MpSoCs while accuracy remains in acceptable level. They can be obtained basically from three sources: stand alone simulators, third party components and created by ADL (Architecture Description Language) based tools.

ADLs' modeling levels are classified into three categories: structural, behavioral, and mixed. Structural or cycle-accurate ADLs describe the processor at a low abstraction level (RTL) with a detailed description of the hardware blocks and their interconnection. These tools, such as *MIMOLA* **[16]**, are mainly targeted for synthesis and not for design space exploration due to their slow simulation speed and lack of flexibility.
On the contrary, behavioral or functional ADLs abstract the microarchitectural details of the processor, and provide a model at the instruction set level. Their low accuracy is compensated by their fast simulation speed. Many languages exist such as *nML* **[17]** and *ISDL* **[18]**.

Therefore, mixed ADLs provide a compromise solution and combine the advantages of both the structural (accuracy) and behavioral (simulation speed) ADLs. It is the best abstraction layer for design space exploration. *EXPRESSION* **[19]**, *MADL* **[20]**, *LISA* **[21]**, and *ArchC* **[22]** are examples of mixed ADLs. A recent type of processor description language called *ArchC* **[23]** is gaining special attention from the research communities **[24]**, **[25]**, **[26]**. *ArchC 2.0* is an open-source ADL, developed by the University of Campinas in Brazil. It generates from processor and

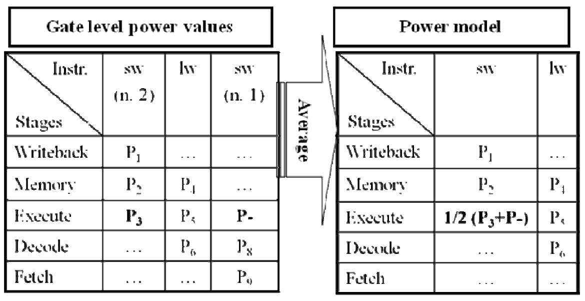
ISA description files, a functional or cycle-accurate ISS in SystemC. The ISS is ready to be integrated with no effort in a complete SoC design based on SystemC **[27]**. In addition, the ISS can be easily deployed in a multiprocessor environment thanks to the interruption mechanism based on TLM, which allows the preemption and migration of tasks between the cores. The main distinction of *ArchC* is its ability to generate a cycle-accurate ISS with little development time. Only the behavior description of the ISA requires accurate description. As for the microarchitectural details, they are generated automatically according to the architecture resource description file. Since *ArchC* is an open-source language, we can modify the simulator generator to produce a processor with customized microarchitectural enhancements, which makes it a great tool for computer architecture research **[29]**. To our knowledge, the processor model cannot be synthesized because it is not yet supported by *ArchC*.

3.2 Power-ArchC Methodology

A Processor designed at Instruction level or RTL has much less details about the design. Hence, to characterize power, the design with same functionality at least at gate-level is required to provide sufficient details. Consequently, the RTL description of the processor is performed manually relatively to the architecture description made in *ArchC*. From that, commercial tools like *PrimeTime* generate a gate level generation, based on a chosen technology library. This explains why the design flow is currently semi-automatic.

Power simulation tools at gate level can provide accurate power consumption of each block, at each clock cycle. Since tools only provide power consumptions of hardware blocks, we designed a parser tool that outputs the average power consumption of each instruction from power and program traces provided by the simulation tool. The parser tracks the power value in the different pipeline stages flowed by the instruction. Based on the data path in each stage of the pipeline, it gathers the power value caused by the execution of one instruction and stores it in a model. Each instruction with same name will only have one entry in the previously defined model, and their different power values are averaged for each stage. For example, let's consider a set of three instructions (2 stores (*sw*) and 1 load (*lw*)) walking in the pipeline as shown in Figure 1 with the power values P_1 to P_9 generated by the gate level tool (other values are not shown for better clarity).

From this example, we show how we build our power model based on sample of instructions that is by computing an average value of gate level power values for same instructions.


Gate level power values

Instr. / Stages	sw (n. 2)	lw	sw (n. 1)
Writeback	P_1	…	…
Memory	P_2	P_4	…
Execute	$\mathbf{P_3}$	P_5	**P-**
Decode	…	P_6	P_8
Fetch	…	…	P_9

Power model

Instr. / Stages	sw	lw
Writeback	P_1	…
Memory	P_2	P_4
Execute	**1/2 (P_3+P-)**	P_5
Decode	…	P_6
Fetch	…	…

Figure 1. An example of generating Power-model

A characterization campaign has to evaluate all of the instructions for different operands and instructions interrelations. Since it is not possible to cover all the possible cases in a reasonable time, only a subset of possibilities is usually considered. In our case, *ILPC* outputs a power model that provides an average total power value (static and dynamic) for each instruction. The flow is illustrated in Figure 2. Chain of tools is shown in black boxes. Input and output files are represented by grey boxes. Section 4 will detail the flow used for MIPS case study.

The power model is next used to back-annotate the ISS generated by *ArchC* with power values for each instruction. The behavior description of the instruction contains now a variable that points to the corresponding instruction in the power model. The ISS is now able to output both instruction and power traces and total consumed power of the executed program.

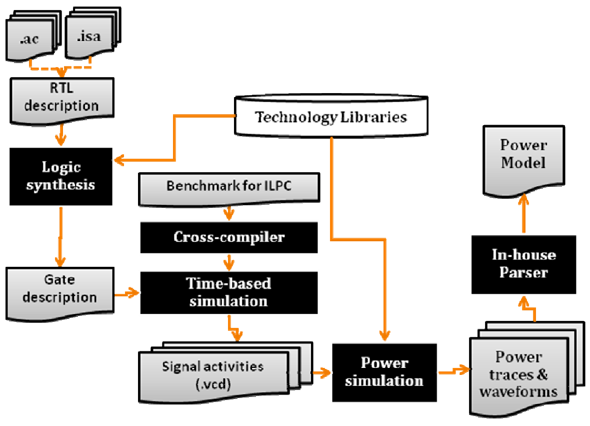


Figure 2. ILPC flow

Basic and modified architecture of ISS is illustrated in Figure 3. From two input description files (ac, isa), the tool *acsim* generates a systemC based ISS that can be easily integrated in a complex MPSoC model. Given that an input power model, the compiler for host machine generates the modified ISS with power capabilities. A new ISS output provides power traces at the end of simulation of a benchmark.

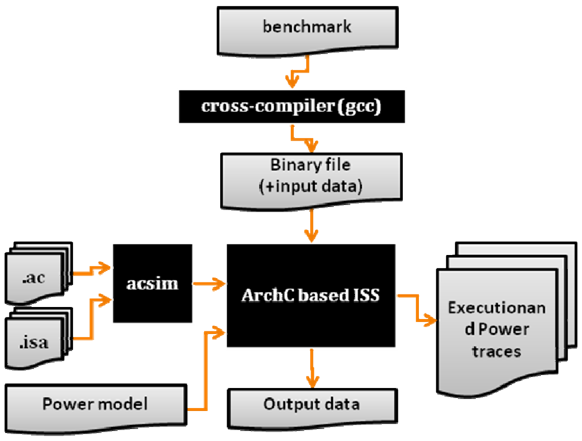


Figure 3. ArchC framework

From that, it is possible to explore the power consumption at high level in a quick fashion. As we experience with performance accuracy at instruction-level, the resulting power accuracy will be obviously lower than the one obtained after synthesis. This is mainly due to the interrelations between the instructions during execution and the value of the operands that both affect the bit toggling activity of the processor micro-architecture and hence, the dynamic power consumption.

One should note that the ILPC can be performed automatically whenever hardware implementation of microarchitecture or technology library changes.

4. Results

We implemented the applied methodology discussed in Section 3 on a MIPS 32-bit processor. We use *ArchC 2.0* to generate the MIPS ISS that supports the full *R3000* ISA. We gathered an open source RTL description of MIPS that is HMC-MIPS **[37, 42]**. It is a project handled by HMC's and Adelaide's Universities to create a MIPS in Verilog language. It includes a 5-stage pipelined processor that mostly supports MIPS ISA. It includes thirty-two general-purpose 32-bit registers and fifty-eight instructions, each 32 bits long. It does not include support for an FPU. It also includes data and program cache memories and a RAM. We applied some modifications in HMC-MIPS to obtain same functionality as from processor described from *ArchC*. We especially deactivated cache memories that are not modeled in MIPS *ArchC*. With *TSMC 45nm* standard cell libraries and Synopsys *Design Compiler v2009.06* **[38]**, we synthesized at gate-level the processor description from HMC-MIPS. After that, we simulated some MiBench benchmark **[8]** execution with Mentor Graphics *Modelsim v6.5b* **[39]** to generate the execution trace of the entire processor. Binary codes are generated with a MIPS cross compiler provided in ArchC framework. A specific link script is made for HMC-MIPS target. Then, Synopsys *PrimeTime PX v2008.12* **[31]** computes a time-based power analysis and reports power traces. Three power models are built from three different ILPC campaigns. In a first ILPC campaign, we use *'Motion'* benchmark **[40]**. Second ILPC campaign is performed with modified version of *'Qsort'* benchmark, which is one of the benchmarks from MiBench and last ILPC campaign is performed with combination of both previous benchmarks. *Motion* and *Qsort* are chosen with small input files due to their simplicity. As an example, the three power models for a subset of MIPS ISA is shown in Table 1.

Table 1. ILPC campaigns of MIPS (power models)

Instr Name	ILPC 1 (in µW)	ILPC 2 (in µW)	ILPC 3 (in µW)
addiu	865.9	953.5	909.7
li	866.6	923.2	894.9
jal	783.1	932.5	857.8
sw	872.2	913.3	892.7
move	867.6	903.6	885.6
lw	849	939.5	894.3
nop	856.3	925.4	890.9
mult	828.6	915.5	872.1
mflo	831.2	845	838.1
j	854.8	913.8	884.3
bgez	899.8	936.6	918.2
sll	866.1	882.9	874.5
addu	871.7	936.3	904.0
slt	868.5	687.3	777.9
bnez	869.7	903.9	886.8
jr	695	918.8	806.9
slti	870.3	989.3	929.8
subu	871.9	915.3	893.6
beqz	866.8	909.9	888.4
negu	908.1	899.9	904.0

ILPC 2 v/s ILPC 1 has an average relative deviation percentage of 6.7%, ILPC 3 v/s ILPC 1 with 3.3%, and ILPC3 v/s ILPC2 with –2.7%. The extremities are with instruction *'jr'* which has maximum relative deviation percentage for ILPC 2 v/s ILPC 1 of 32.2% and instruction *'negu'* has minimum for ILPC 2 v/s ILPC 1 of -0.9%.

From the three ILPC campaigns, Table 2 shows the energy consumption of 8 MiBench benchmarks whereas Table 3 shows the average power consumption. MiBench benchmarks with six suites, each suite targeting a specific area of the embedded market.

The six categories are Automotive and Industrial Control, Consumer Devices, Office Automation, Networking, Telecommunications. The difference in average power for different benchmarks is due to instruction distribution, memory behavior, and available parallelism. The difference in total energy is mainly due to the complexity and the number of instructions (hence the execution time of a benchmark) and also due to interactions between instructions.

Table 2. Total energy in µJ of benchmarks vs. ILPC campaigns

Benchmarks	ILPC 1 (in µJ)	ILPC 2 (in µJ)	ILPC 3 (in µJ)
Bitcount	251.8	349.2	339.7
Qsort	80.6	127.6	124.0
Susan	19.4	30.0	29.3
Jpeg	216.9	250.7	244.3
Gsm	223.6	235.6	228.9
Stringsearch	1.8	2.5	2.4
Rijndael	172.1	264.0	257.3
Patricia	1672.3	2549.5	2487.1

This information is useful to designers to consider the effect of these design constraints on power consumption and eventually may result in reducing discharge rate of batteries in portable computing devices.

For all benchmarks 'Qsort' has the maximum relative deviation percentage for ILPC 2 v/s ILPC 1 of 58.2%, whereas 'gsm' has the minimum of 5.3%. Table 2 and Table 3 show the difference between different campaigns. The difference is quite visible, it is because of the use of two small campaigns that explain the need of more intensive simulations for more campaigns and for various operand data to achieve better accuracy.

Table 4 shows the energy consumption of both *Motion* and *Qsort* benchmarks at instruction-level and gate-level. For each ILPC campaign, we compute the relative energy error. For *Motion* benchmark, the energy value in ILPC1 column is equal to gate level value. Actually, ILPC1 at gate level is performed by the execution of this benchmark (with same input data).

Table 3. Average Power in µW of benchmarks vs. ILPC campaigns

Benchmarks	ILPC 1 (in µW)	ILPC 2 (in µW)	ILPC 3 (in µW)
Bitcount	552.4	765.9	745.2
Qsort	559.6	885.3	860.6
Susan	561	867.8	848.6
Jpeg	736	850.6	829
Gsm	684.7	721.3	700.8
Stringsearch	666.2	905.4	878.9
Rijndael	510.4	783.1	763.2
Patricia	578.2	881.5	859.9

However, a relative deviation of 2.31% appears instead of 0. This difference is due to two reasons: one is the difference in linker scripts and second is due to the HMC-MIPS difference with the MIPS-ISA for the *'move'* instruction. For ILPC2 and ILPC3, a relative deviation of resp. 1.3% and 6.1% appear. In a similar way, *Qsort* benchmark may display an identical value in gate-level and ILPC2 columns. As explained above, a difference appears for the same reasons. With ILPC1 and ILPC3 based power models, we measure a relative deviation of resp. 21% and 0.5%. Additional simulations at instruction-level with different operand values will be performed for a better validation.

Table 4. Total energy in µJ of MIPS at instruction level vs. gate-level for three ILPC campaigns

Bench-Mark	Energy at gate level (in µJ)	Energy at instruction-level		
		ILPC 1 (in µJ)	ILPC 2 (in µJ)	ILPC 3 (in µJ)
Motion	0.2664	0.2726	0.2628	0.2827
Qsort	0.0438	0.0426	0.0344	0.0441

As an additional feature, Power model also includes the power values of each instruction per pipeline stage. Figure 4 shows the total energy of each MIPS pipeline for each benchmark. It is specific to the architecture, but it can be easily seen which block is consuming more energy and which ones consume less. This information is very important to design more efficient architecture and to study the impact of applications on the energy consumption.

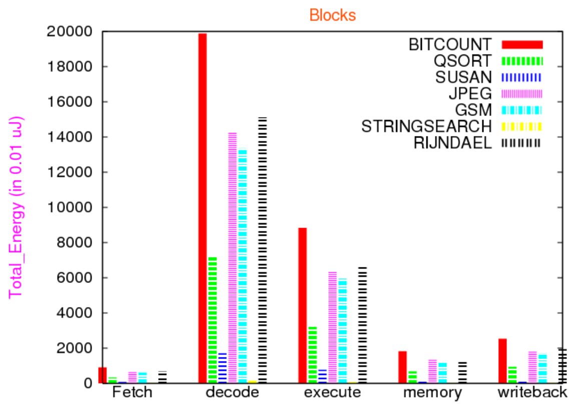


Figure 4. Power-ArchC feature: Energy consumption of MIPS pipeline at instruction level

5. Conclusion

In this paper, we have shown a methodology to explore power and energy consumption for an ISS generated by *ArchC*. The originality of our work is that it is the first work that incorporates power capabilities in the *ArchC* framework and so it can provide immediate power estimation in MPSoC with the execution of a benchmark. The limitations of this methodology are:

- To reduce the time of obtaining power model, we take into account only a subset of all the possible operands and instructions interrelations to characterize power consumption, and
- There is no tool at present, able to translate an *ArchC* description to RTL. Hence, for different processor architecture than MIPS, we need ISA description in *ArchC* and in RTL with same functionalities.

Although, there seems to be an update for *ArchC* in future that may remove the second limitation **[41]**. For first limitation, it can be overcome by performing intensive simulations for benchmarks that use different operands and instruction order. These results can help improving the design performance and implementing specific algorithms for optimizing power consumption, controlling temperature and improving reliability at early design stage.

6. References

[1] J. Rabaey and M. Pedram (Editors), *Low Power Design Methodologies*. Kluwer Academic Publishers, Norwell, MA, 1996.
[2] E. Macii, M. Pedram, and F. Somenzi, .High-level power modeling, estimation, and optimization,. in *Proc. Design Automation Conf.*, pp. 504.511, June 1997.
[3] A. Raghunathan, N. K. Jha, and S. Dey, *High-level Power Analysis and Optimization*. Kluwer Academic Publishers, Norwell, MA, 1998.
[4] Coburn, J., Ravi, S., and Raghunathan, A. 2005. Power emulation: a new paradigm for power estimation. In *Proceedings of the 42nd Annual Design Automation Conference* (Anaheim, California, USA, June 13 - 17, 2005). DAC '05. ACM, New York, NY, 700-705.
[5] T.H. Krodel. PowerPlay - Fast Dynamic Power Estimation Based on Logic Simulation. IEEE International Conference on Computer Aided Design, pp. 96-100, Oct. 1991.
[6] R. Tjarnstorm. Power Dissipation Estimate by Switch Level Simulation. IEEE symposium on Circuits and Systems, pp. 881-884, 1989.
[7] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. Computer Architecture, 2000. Proceedings of the 27th International Symposium.
[8] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown. MiBench: A free, commercially representative embedded benchmark suite. Workload Characterization. 2001 IEEE International Workshop. WWC-4.
[9] EECS UMICH. "Sim-Panalyzer," http://www.eecs.umich.edu/~panalyzer/
[10] Mehta, H., Owens, R. M., and Irwin, M. J. 1996. Instruction level power profiling. In Proceedings of the Acoustics, Speech, and Signal Processing, 1996. on Conference Proceedings., 1996 IEEE international Conference - Volume 06 (May 07 - 10, 1996). ICASSP. IEEE Computer Society, Washington, DC, 3326-3329.
[11] Tiwari, V., Malik, S., Wolfe, A., and Lee, M. T. 1996. Instruction Level Power Analysis and Optimization of Software. In Proceedings of the 9th international Conference on VLSI Design: VLSI in Mobile Communication (January 03 - 06, 1996). VLSID. IEEE Computer Society, Washington, DC, 326.
[12] Krintz, C., Wen, Y., and Wolski, R. 2004. Application-level prediction of battery dissipation. In Proceedings of the 2004 international Symposium on Low Power Electronics and Design (Newport Beach, California, USA, August 09 - 11, 2004). ISLPED '04. ACM, New York, NY, 224-229.
[13] Coware company available at http://www.coware.com, May 2003.
[14] Wu-Chun Feng (October 2003). "Making a case for Efficient Supercomputing". ACM Queue 1 (7).
[15] Nguyen, H. T., Chatterjee, A., and Roy, R. K. 1998. Activity Measures for Fast Relative Power Estimation in Numerical Transformation for Low Power DSP Synthesis. J. VLSI Signal Process. Syst. 18, 1 (Jan. 1998), 25-38.

[16] Rainer Leupers and Peter Marwedel. Retargetable code generation based on structural processor descriptions. In *In Design Automation for Embedded Systems,* pages 1–36. Kluwer Academic Publishers, 1998.

[17] A. Fauth, J. Van Praet, and M. Freericks. Describing instruction set processors using nml. In EDTC '95: Proceedings of the 1995 European conference on Design and Test, page 503, Washington, DC, USA, 1995. IEEE Computer Society.

[18] G. Hadjiyiannis, S. Hanono, and S. Devadas. Isdl: An instruction set description language for retargetability. In Design Automation Conference, 1997. Proceedings of the 34th, pages 299–302, Jun 1997.

[19] Ashok Halambi, Peter Grun, Vijay Ganesh, Asheesh Khare, Nikil Dutt, and Alex Nicolau. Expression: a language for architecture exploration through compiler/simulator retargetability. In DATE '99: Proceedings of the conference on Design, automation and test in Europe, page 100, New York, NY, USA, 1999. ACM.

[20] Wei Qin, Subramanian Rajagopalan, and Sharad Malik. A formal concurrency model based architecture description language for synthesis of software development tools. In LCTES '04: Proceedings of the 2004 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems, pages 47–56, New York, NY, USA, 2004. ACM.

[21] Stefan Pees, Andreas Hoffmann, Vojin Zivojnovic, and Heinrich Meyr. Lisa—machine description language for cycle-accurate models of programmable dsp architectures. In DAC '99: Proceedings of the 36th annual ACM/IEEE Design Automation Conference, pages 933–938, New York, NY, USA, 1999. ACM.

[22] M. Bartholomeu G. Araujo C. Araujo R. Azevedo, S. Rigo and E. Barros. The ArchC Architecture Description Language and Tools. Parallel Programming, 33(5):453–484, 2005.

[23] S. Rigo, G. Araujo, M. Bartholomeu, and R. Azevedo. Archc: a systemc based architecture description language. In Proc. 16th Symposium on Computer Architecture and High Performance Computing SBAC-PAD 2004, pages 66–73, 2004.

[24] G. Beltrame, C. Bolchini, L. Fossati, A. Miele, and D. Sciuto. Resp: A non-intrusive transaction-level reflective mpsoc simulation platform for design space exploration. In Proc. Asia and South Pacific Design Automation Conference ASPDAC 2008, pages 673–678, 2008.

[25] M. R. de Schultz, A. K. I. Mendonca, F. G. Carvalho, O. J. V. Furtado, and L. C. V. Santos. Automatically-retargetable model-driven tools for embedded code inspection in socs. In Proc. 50th Midwest Symposium on Circuits and Systems MWSCAS 2007, pages 245–248, 5–8 Aug. 2007.

[26] N. Kavvadias and S. Nikolaidis. Elimination of overhead operations in complex loop structures for embedded microprocessors. 57(2):200–214, Feb. 2008.

[27] Open SystemC Initiative:. http://www.systemc.org.

[28] C. Araujo, M. Gomes, E. Barros, S. Rigo, R. Azevedo, and G. Araujo. Platform designer: An approach for modeling multiprocessor platforms based on systemc. Design Automation for Embedded Systems, 10(4):253–283, 2005.

[29] Sandro Rigo, Marcio Juliato, Rodolfo Azevedo, Guido Ara´ujo, and Paulo Centoducatte. Teaching computer architecture using an architecture description language. In WCAE '04: Proceedings of the 2004 workshop on Computer architecture education, page 6, New York, NY, USA, 2004. ACM.

[30] Slater, M. 1992. MIPS R3000 system design. In *A Guide To RISC Microprocessors*, M. Slater, Ed. Academic Press Professional, San Diego, CA, 99-106.

[31] Gordon, Y. June 2006. Expanding the Synopsys PrimeTime® Solution with Power Analysis.

[32] Ye, W., Vijaykrishnan, N., Kandemir, M., and Irwin, M. J. 2000. The design and use of simplepower: a cycle-accurate energy estimation tool. In *Proceedings of the 37th Annual Design Automation Conference* (Los Angeles, California, United States, June 05 - 09, 2000). DAC '00. ACM, New York, NY, 340-345.

[33] Sinha, A. and Chandrakasan, A. P. 2001. JouleTrack: a web based tool for software energy profiling. In *Proceedings of the 38th Annual Design Automation Conference* (Las Vegas, Nevada, United States). DAC '01. ACM, New York, NY, 220-225.

[34] Senn, E., Laurent, J., Julien, N., and Martin, E. 2005. Softexplorer: estimating and optimizing the power and energy consumption of a C program for DSP applications. *EURASIP J. Appl. Signal Process.* 2005 (Jan. 2005), 2641-2654.

[35] Klein, F., Araujo, G., Azevedo, R., Leao, R., and dos Santos, L. C. 2007. A multi-model power estimation engine for accuracy optimization. In *Proceedings of the 2007 international Symposium on Low Power Electronics and Design* (Portland, OR, USA, August 27 - 29, 2007). ISLPED '07. ACM, New York, NY, 280-285.

[36] C. Araujo, M. Gomes, E. Barros, S. Rigo, R. Azevedo, and G. Araujo. Platform designer: An approach for modeling multiprocessor platforms based on SystemC. *Design Automation for Embedded Systems*, Vol. 10:253--283, 2007.

[37] "Google Code hmc-mips," http://code.google.com/p/hmc-mips/

[38] Synopsys, Design Compiler, http://www.synopsys.com/Tools/Implementation/RTLSynthesis/Pages/DesignCompiler2010-ds.aspx

[39] ModelSim: http://www.model.com/

[40] L. M. Po and W. C. Ma, "A novel four-step search algorithm for fast block motion estimation," IEEE Trans. Circuits Syst. Video Technol., vol. 6, no. 3, pp. 313-317, Jun. 1996.

[41] Goto, S., DVCon 2010. ADL Synthesis using ArchC. www.nascug.org/events/12th/nascug12_samuel_goto.pdf

[42] Pinckney, N., Barr, T., Dayringer, M., McKnett, M., Jiang, N., Nygaard, C., Harris, D. M., Stanley, J., and Phillips, B. 2008. A MIPS R2000 implementation. In Proceedings of the 45th Annual Design Automation Conference (Anaheim, California, June 08 - 13, 2008). DAC '08. ACM, New York, NY, 102-107.