An Industrial Case Study of the ARM926EJ-S Power Modeling

Hyunsuk Kim[1], Seokhoon Kim[1], Ikhwan Lee[1], Sungjoo Yoo[1], Eui-Young Chung[2], Kyu-Myung Choi[1], Jeong-Taek Kong[1], Soo-Kwan Eo[1]
[1]CAE center, System LSI division, Semiconductor Business, Samsung Electronics, Co. Ltd.
San #24 Nongseo-Ri, Giheung-Eup, Youngin-City, Gyeonggi-Do, Korea, 449-711
[2]School of Electrical and Electronic Engineering, Yonsei University
134 Sinchon-dong, Seodaemun-gu, Seoul, Korea, 120-749

Abstract – *In this work, our goal is to develop a fast and accurate power model of the ARM926EJ-S processor in the industrial design environment. Compared with existing work on processor power modeling which focuses on the power states of processor core, our model mostly focuses on the cache power model. It gives more than 93% accuracy and 1600 times speedup compared with post-layout gate-level power estimation. We also address two practical issues in applying the processor power model to the real design environment. One is to incorporate the power model into an existing commercial instruction set simulator. The other is the re-characterization of power model parameters to cope with different gate-level netlists of the processor obtained from different design teams and different fabrication technology.*

Keywords: power estimation, processor, ARM926EJ-S, cache, sequential / non-sequential access, fill buffer, instruction set simulator, re-characterization

1 Introduction

Embedded software design is getting more and more attention as software (SW) complexity increases faster than hardware (HW) complexity [1]; thus, SW cost starts to dominate total chip design cost [2]. SW dominates power consumption as well as system performance. In particular, handheld mobile devices (e.g., cell phone, PDA, PMP, and MP3 player) require SW running on the devices to consume minimum power.

Low power SW design technology covers a wide range of research area, e.g., instruction scheduling [10], dynamic voltage scaling [11], and code transformation to reduce off-chip memory accesses [12]. In reality, SW designers often apply manual code optimization for both performance and power. Thus, the design space of low power SW is huge in terms of design technology and ad-hoc manual optimization.

It is crucial to allow SW designers to explore the huge SW design space to achieve low power design. To do that, we need accurate and fast methods of estimating the power consumption of SW running on the target processor.

In this work, our goal is to develop a fast, but accurate power model of the ARM926EJ-S processor in the industrial design environment. The industrial design environment is different from the academic arena in two aspects. First, designers often resort to commercial simulators and tools. Thus, the power model needs to be incorporated into their existing simulators or tools. Second, there is a need to re-characterize the power model parameters to deal with different gate-level netlists of the processor obtained from different design teams and different fabrication technology[1]. In order to apply the power model to a real design environment, we need to resolve these two issues.

Compared with existing work on processor power modeling which focuses on the power states of processor core, our model mostly focuses on the cache power model. It is because cache activities dominate the variation of processor power consumption and our initial goal of power estimation accuracy (90% in average power) justified a simple power model of the processor core.

Experiments show that the presented power model gives more than 93% accuracy and 1600 times speedup compared with power simulation in post-layout gate-level.

The paper is organized as follows. Section 2 summarizes related work. Section 3 explains the presented power model of the ARM926EJ-S. Section 4 addresses the issue of integrating the power model into a commercial instruction set simulator. Section 5 presents an environment to allow for the re-characterization of the power model parameters. Section 6 gives experimental results. Section 7 concludes the paper.

2 Related Work

Numerous studies have proposed various processor power modeling techniques. The first work on processor power modeling was reported in [6]. Their model quantified instruction base energy and various inter-instruction energy effects to enable fast software energy estimation. Wattch [4] and SimplePower [5] are two well-known power estimation tools in academia.

A power model tailored for the Intel XScale processor was proposed in [7]. Their power model is based on module

[1] For instance, 130nm technology may have different versions such as high speed, generic, low power and different libraries such as normal CMOS, MTCMOS, etc. Migration from 130nm to 90nm or 65nm adds another dimension of different types of netlists.

activities, where each module has its power equation embedded in Sim-XScale simulator. The power equations were constructed using transistor level schematics of functional units and a high-level view of transistor gate and drain capacitances.

A software power estimation tool, JouleTrack, was presented in [9]. They proposed power characterization methodology that avoids explicit power characterization for each differentiated instruction class.

A power model for the Philips PR1900 processor was proposed in [8]. Their elaborate power model is instruction-based similar to [6] and the base power values was obtained using their in-house gate-level power estimation tool.

Compared with the aforementioned studies, our work is unique in that we resolve practical issues encountered in applying power modeling to the ARM926EJ-S processor in the industrial design environment. More specifically, we resolve unique issues such as memory compiler usage (in Section 3), cache-oriented power modeling (in Section 3), integration of the power model into a commercial instruction set simulator (in Section 4), and re-characterization of the power model parameters (in Section 5).

3 ARM926EJ-S Processor Power Model Development

High-level power modeling involves three major steps: Defining power states, characterizing power values per state, and annotating the simulator with the power values. In our work, our goal is to achieve at least 90% of power estimation accuracy compared with gate-level power estimation using the post-layout netlist.

3.1 ARM926EJ-S architecture

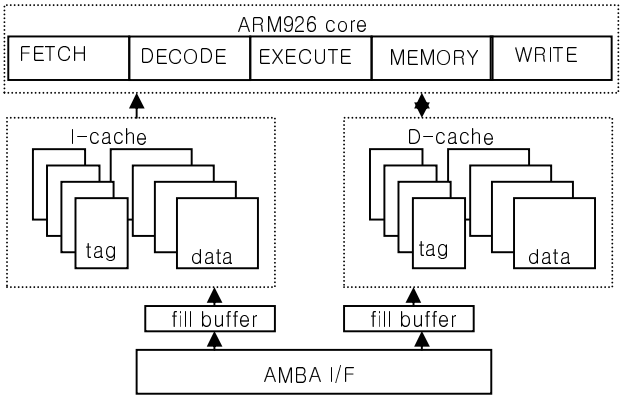

Figure 1 ARM926EJ-S architecture.

The ARM926EJ-S processor has a five stage pipelined datapath and a Harvard cache architecture as shown in Figure 1. The caches are four-way set associative, with a cache line length of eight words per line. The size of the caches can be from 4KB to 128KB. The ARM926EJ-S processor also has a fill buffer (FB) that keeps the most recently fetched cache line.

Sequential / non-sequential cache accesses

In the ARM926EJ-S processor, any instruction that modifies the program counter (such as a branch, or 'MOV pc, r0') causes a non-sequential instruction accesses on the next cycle. An instruction access by 'PC increment by 4' that crosses the cache line boundary also causes a non-sequential access. In Figure 2 (a), a non-sequential (NS) access causes all four cache tag memories and data memories to be accessed along with the fill buffer. Whereas a sequential access (SEQ) causes only the data memory where the data is located is accessed as in Figure 2 (b). In Figure 2 (c), if the data is accessed from the fill buffer, there in no access to the cache.

For data caches, load multiple (LDM) and store multiple (STM) instructions support sequential accesses. LDR and STR instructions incur non-sequential accesses.

3.2 ARM926EJ-S power states

The ARM926EJ-S processor is mainly composed of the processor core (mostly consisting of logic) and memory cells (i.e. instruction and data caches, fill buffers, MMU, etc.) as shown in Figure 1. We separate the processor power model into two parts: Processor core model and cache model. This separation comes from two observations. One is that caches can be configured differently (in terms of size, associativity, etc.) for various applications. Thus, one single model will not give an accurate estimation. The other observation is that the power consumption of caches gives a large variation. In the ARM926EJ-S processor, the cache power consumption ranges from 3% up to 60% of total power. Therefore, we decide to model the core logic block and cache memory separately.

3.2.1 Processor core: Two simple power states

We observe that the core logic can be in one of the two states: Busy state and idle state (stalled by interlocks). There are numerous studies on processor power modeling, where more complex instruction level power states are identified [6,7,8]. However, in our work, we find that the two-state core power model gives more than 95% of the core power estimation accuracy for all of our benchmarks. On the other hand, one state model performs very poorly with its accuracy level of less than 70% for some benchmarks. Thus, we adopt the two-state power model for the processor core.

3.2.2 Activity-based coarse-grain cache power model

Most of the previous work on cache power modeling has exploited circuit-level information such as bit line and word line capacitive loads to generate flexible cache power models [4, 5, 7]. In industry, cache memories use memory compiler-generated SRAMs, where power values for each module are also provided for each type of read and write

access. Thus, our cache power is modeled as a sum of power values for all accessed SRAM modules. For SRAM modules not accessed during the cycle, their static power values are added.

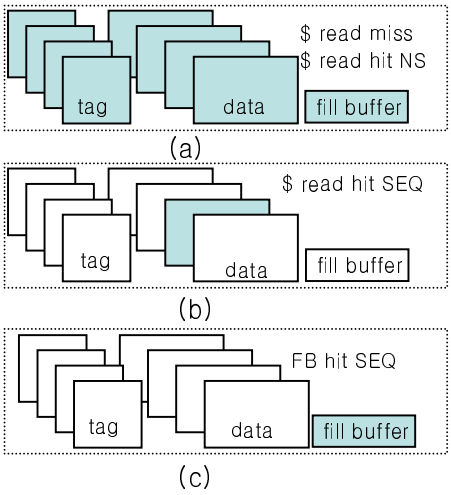

Figure 2 ARM926EJ-S cache activity patterns.

Power modeling for sequential/non-sequential accesses
The ARM926EJ-S cache access behavior can be categorized into three different types as shown in Figure 2. In power perspective, a non-sequential access consumes more than four times of power than a sequential access, since the cache power is the sum of dynamic power of all activated modules (tag memories and data memories) and static power of inactive modules. In Figure 3, the CSN (Chip Select Negative) signals for four cache ways are shown with the total power consumption graph measured using our in-house gate-level power simulator for the short code segment at top of the figure. When the four CSN signals are active (four ways are accessed altogether), the access is non-sequential, whereas if only one of the four ways is accessed, the access is sequential. The figure dictates that in the ARM926EJ-S caches, non-sequential accesses and sequential accesses should be differentiated for accurate power estimation.

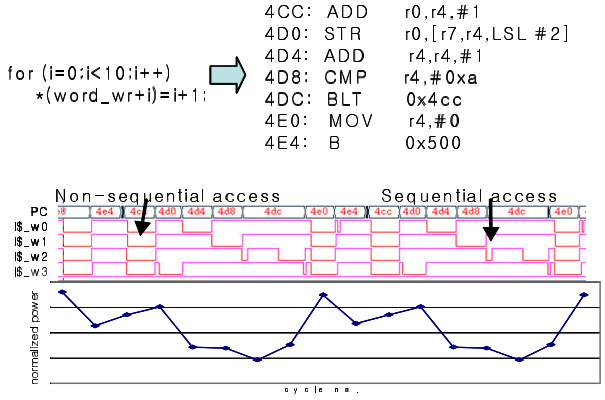

Figure 3 Comparison of power consumption between sequential and non-sequential accesses.

Table 1 lists our identified data cache states and their corresponding module activities and power equations. In the table, *Tr* (*Tw*) and *Dr* (*Dw*) represent module power numbers for *Tag read* (*Tag write*) and *Data read* (*Data write*), respectively, obtained from our in-house memory compiler. The states are identical for the instruction cache except that there is no cache write hit or miss states. In this work, we ignore the power consumed by fill buffers.

Power modeling for fill buffer accesses
Instructions and data are accessed from the fill buffer until it is evicted to the cache in two cycles (as shown 1st write-back and 2nd write-back in Table 1) by the following cache line fetched in from the bus. Instruction fill buffer (I-FB) hit counts accounts for approximately 10% of the instruction cache hit counts in our *dhrystone* benchmark. If an instruction fill buffer hit is encountered and the PC increments by 4, then it is I-FB sequential read, where negligible amount of power is consumed by the fill buffer. Therefore, it should be distinguished if the data is read from the cache or fill buffer to estimate power accurately.

Power modeling for data write accesses
Data cache write event takes at least three cycles. Four tags are first matched to find if it is a hit or a miss. If it is a hit, the data is written via a write buffer. We did not consider the power consumption of the write buffer since it is considered to be negligible. If the access is a miss, it is written externally.

Table 1 Activity-based cache power model.

Cache states	Module activity	Power Equation
sequential (cache) read	1 data read	Dr
non-sequential (cache) read / read miss	4 tag reads and 4 data reads	Tr*4 + Dr*4
Data cache write hit	4 tag reads, 1 tag write and 1 data write	Tr*4 +Tw+Dw
Data cache write miss	4 tag reads	Tr*4
FB -> cache write (1st write-back)	1 tag write and 4 data writes	Tw+Dw*4
FB -> cache write (2nd write-back)	4 data writes	Dw*4
sequential FB read	-	-

4 Inferring Cache Power States

The module activity information shown in Table 1 is not available in our instruction set simulator. The simulator reports only cache miss, cache hit, and fill buffer hit events without the information on sequential and non-sequential accesses. To obtain the activity information, distinctive cache power states need to be inferred from the available state information of the simulator at run-time.

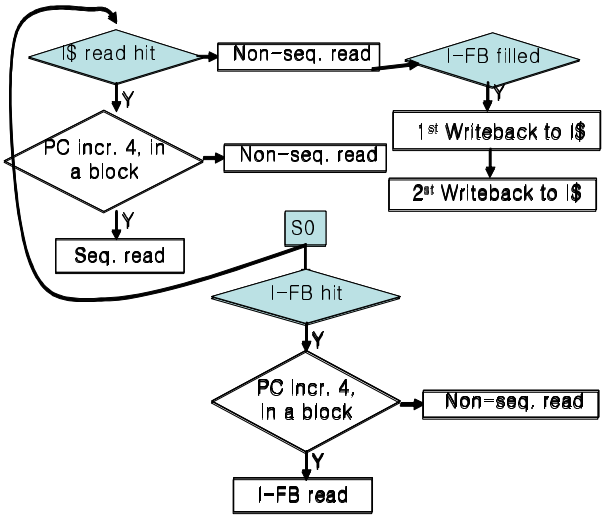

Figure 4 Inference flow diagram of sequential / non-sequential / FB accesses for the instruction cache.

To infer the required cache activity information, we implement the steps in the flow diagram depicted in Figure 4 in the instruction set simulator for instruction accesses (a similar flow diagram for data caches is employed in our work). While observing the PC update history, we use the statistics such as cache hit or miss, and fill buffer hit or miss, provided by the instruction set simulator to infer the instruction cache power state. For instance, if a cache read hit is reported while the PC is incremented by 4 inside a cache line, it is in sequential read state. A fill buffer hit event reported by the simulator can be in one of the two power states as shown in Figure 2 (a) (fill buffer access in non-sequential access) and Figure 2 (c) (fill buffer access in sequential access). Similar to the cache hit event, if the fill buffer hit event is reported while the PC is incremented by 4 inside a cache line, it is in sequential fill buffer read state.

5 Power Re-characterization Flow

Power consumption is a complex function of many parameters. Depending on the quality of implementation, the same RTL can result in very different power values in the gate-level netlist. For example, two of our sample designs of the ARM926EJ-S show as much as twice power difference at the same frequency level, even though they are implemented with the same technology library. This implies that '*characterize once*' approach might not hold true in real applications.

In general, power characterization in gate level proceeds as follows: (1) Obtain the signal toggle information from gate level simulation, (2) estimate the gate-level power from the toggle information using power libraries, and (3) calculate per-state power values using the estimated power information. If the power characterization is performed manually for each different gate-level netlist, it will be long, tedious, and error-prone task.

To reduce the characterization efforts, we set up an automated characterization flow as shown in Figure 5, where designers can characterize power values repeatedly without investing much effort. The characterized power values are simply read by our simulator annotated with the power model explained in Section 4 to produce software power profiles. Note that the power model itself does not need any modification. We find that the power model itself is valid for different implementations of the same RTL.

Figure 5 shows our power characterization flow. We first build a gate-level and RTL co-simulation template, where an RTL testbench with a simple bus and memory module drives the simulation with the ARM926EJ-S gate level netlist of interest to generate the cycle-by-cycle signal toggle information as well as signal traces to infer the power states, using *dhrystone* benchmark. The toggle information is then fed into our in-house gate level power estimation tool to generate cycle-by-cycle power values. The per-state power value is obtained by averaging the estimated cycle-by-cycle power values. All the aforementioned steps are performed automatically without any user intervention. The obtained per-state power value is finally annotated into our power simulator. We use the characterization flow to obtain the core power states in our power model. Note that the cache power model is activity-based and its SRAM module power value is provided by our memory compiler as explained in Section 3.

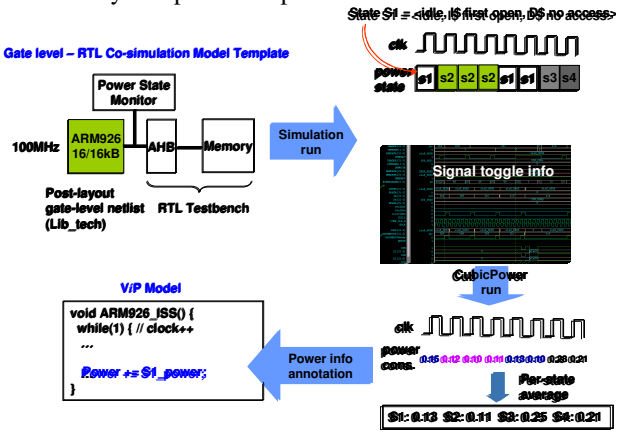

Figure 5 Our power characterization flow.

6 Experiments

Table 2 lists the characteristics of five benchmarks used in our experiments. The Figure 6 shows 93%~98% of average power estimation accuracy for the five benchmarks. Figure 7 shows cycle-by-cycle estimation result for a short code segment. It can be seen that the estimated power values closely track the power values measured in gate level. The cycle-by-cycle error is 17% on average. Regarding the power estimation speed of our simulator, it performs approximately 1600 times faster

than gate-level estimation.

Table 2 Benchmarks.

Benchmark	Code Size	Simulation Cycle Counts
dhrystone	49KB	12068
cavity_detection	39KB	106138
adpcm encoder	36KB	101358
fft	96KB	321537
h264 enc (me intpel only)	714KB	1896639

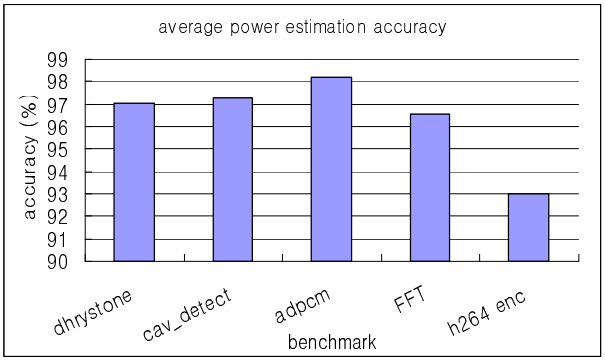

Figure 6 Average power estimation accuracy.

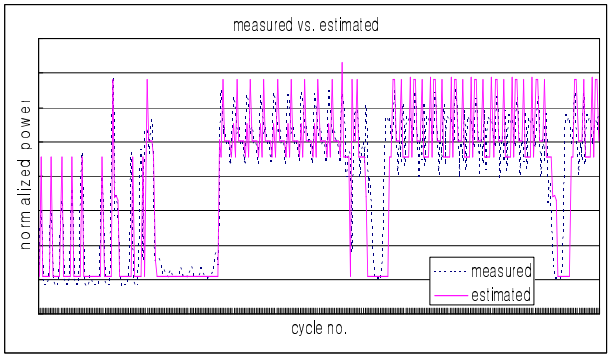

Figure 7 Cycle-by-cycle estimation accuracy.

7 Conclusions

In this paper, we proposed a fast and accurate power model for the ARM926EJ-S processor. The processor core is modeled in two power states, namely, busy state and idle state. The cache model is a coarse-grained activity model. We model power distinctive cache states based on its access behavior. Each power state is inferred by the instruction simulator at run-time using the cache events provided by the simulator.

We also presented the power characterization flow with which each design team can adapt the model to its own implementation of the processor without much effort.

Our experiments report more than 93% of average power estimation accuracy and closely track the cycle-by-cycle power trend.

Our future work includes applying the technique presented in this paper to other processors such as DSPs and the ARM1176 processor.

Acknowledgments

We thank Dr. Joonhwan Yi from TN Business, Samsung Electronics, and Tom Miller from Sequence for their interests and helpful comments during the course of this project.

References

[1] MEDEA+ EDA Roamap, 2003. http://www.medeaplus.org.

[2] International Technology Roadmap for Semiconductor (ITRS), http://public.itrs.net/Files/2003ITRS/Home2003.htm.

[3] ISO/IEC 14496-10 and ITU-T Rec. H.264, Advanced Video Coding, 2003.

[4] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," in *ISCA'00*, pages 83-94, 2000.

[5] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "The design and use of simplepower: a cycle-accurate energy estimation tool," in *DAC'00*, pages 340 – 345, 2000.

[6] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software:a first step towards software power minimization," *IEEE Trans. on VLSI systems*, pages 437-445, December 1994.

[7] G. Gontreras, M. Martonosi, J. Peng, R. Ju, and G.-Y. Lueh, "XTREM: a power simulator for the Intel XScale® core," in *LCTES'04*, pages: 115 – 125, 2004.

[8] A. Sama, M. Balakrishnan, and J. F. M. Theeuwen, "Speeding up Power Estimation of Embedded Software." in *ISLPED'00*, pages 191-196, 2000.

[9] A. Sinha, and A. Chandrakasan, "JouleTrack: a web based tool for software energy profiling," in *DAC'01*, pages: 220 - 225 , 2001.

[10] M. Lee, *et al.*, "Power Analysis and Low-Power Scheduling Techniques for Embedded DSP Software," In *Proc. ISSS*, 1995.

[11] J. Seo, *et al.*, "Profile-based Optimal Intra-task Voltage Scheduling for Hard Real-Time Applications," in *Proc. DAC*, 2005.

[12] DTSE Methodology, http://www.imec.be/design/dtse/methodology.shtml.