System-Level Techniques for Estimating and Reducing Energy Consumption in Real-Time Embedded Systems

Ishihara, Tohru
System LSI Research Center, Kyushu University

Goudarzi, Maziar
System LSI Research Center, Kyushu University

出版情報：International SoC Design Conference, pp.67-72, 2007-10
バージョン：
権利関係：

System-Level Techniques for Estimating and Reducing Energy Consumption in Real-Time Embedded Systems

Tohru Ishihara **Maziar Goudarzi**
System LSI Research Center, Kyushu University, Fukuoka, Japan
{ishihara,goudarzi}@slrc.kyushu-u.ac.jp

Abstract - *Energy consumption is a fundamental barrier in taking full advantage of today and future semiconductor manufacturing technologies. We present our recent research activities and results on estimating and reducing dynamic and static energy under real-time constraints in embedded systems. This includes techniques and tools for (i) estimating instantaneous energy consumption of embedded processors during an application execution, (ii) reducing energy consumption by optimally mapping functions and data items to the scratch-pad memory (SPM), the cacheable, and non-cacheable memory regions of the processor memory space, (iii) reducing the energy consumption of SPM by partitioning it into two sections with different dynamic vs. static power dissipations, (iv) reducing leakage energy in instruction cache memories by taking advantage of value-dependence of SRAM leakage due to within-die V_{th} variation, (v) choosing higher threshold voltage and compensating the delay-violating cache-lines by additional cache ways, and (vi) reducing energy of the logic-part of processor cores by statically implementing multiple same-ISA cores with different energy and performance characteristics.*

Keywords: Embedded system, power estimation, power reduction, dynamic power, static power, scratch-pad memory, cache memory, compiler optimization.

1 Introduction

There is a wide consensus that the energy consumption is a fundamental barrier in taking full advantage of today and future semiconductor manufacturing technologies. We present our recent research activities and results in three categories: estimating software power consumption, reducing energy of the memory subsystem, and dynamically managing the energy of the logic core of the processor. Firstly, we present a technique to estimate instantaneous energy consumption of embedded processors during an application execution. We train a per-processor energy-model which receives statistics from the processor instruction-set simulator (ISS) and gives the instantaneous energy consumption. Secondly, we reduce energy consumption by optimally mapping functions and data items to the scratch-pad memory (SPM), the cacheable, and non-cacheable memory regions of the processor memory space. Thirdly, to reduce the energy consumption of SPM, we partition it into two sections: one section favors more frequently accessed items by providing low-dynamic-energy SRAM cells, while the other section reduces leakage energy by using higher threshold voltage (V_{th}). Both of these sections can be accessed at the same latency, which makes it easy to integrate this SPM into off-the-shelf processor IPs. Fourthly, in order to reduce energy of the logic-part of processor cores, we statically implement multiple same-ISA cores with different energy and performance characteristics. Power management software dynamically selects one of the cores based on the criticality of the task and the proximity of the deadline. Fifthly, we reduce leakage energy in instruction cache memories by taking advantage of value-dependence of SRAM leakage due to within-die V_{th} variation. We propose techniques to better match the instructions to the less-leaky state of their corresponding cache cells. Finally, we reduce leakage energy of caches by using higher V_{th}. Since a few cache-lines will then violate the original cache access latency due to the within-die delay variation of SRAM cells, we replace them with one/two redundant cache-ways so as to keep the cache capacity, latency, and yield intact.

In the rest of this paper, in Section 2 we present the instantaneous power estimation technique. Section 3 provides various techniques to reduce power consumption of the memory subsystem and Section 4 presents our technique for power reduction of the core logic part. Finally, Section 5 summarizes and concludes the paper.

2 Software Power Estimation

This section shows an overview of our energy characterization tool which helps designers in developing a fast and accurate energy model for a target processor-based system. We use a linear model for energy estimation and find the coefficients of the model using multiple linear regression analysis. For more detailed information of our tool see [1].

2.1 Energy Characterization

The energy consumption of a processor can be estimated using the following linear formula,

$$E_{estimate} = \sum_{i=0}^{N} c_i \cdot P_i \qquad (1)$$

where P_i's, c_i's and N are the parameters of the model, the corresponding coefficients and the number of parameters, respectively. The first step for the modeling is to find P_i's required for estimating the energy consumption of the target processor system. The P_i's should be parameters whose values can be easily

obtained using a fast simulator like an ISS. For example, P_i's can be the number of load and store instructions executed, the number of cache misses, etc. Once the required set of parameters is obtained, the next step is to find a training bench for the energy characterization. Note that the number of cycles simulated for the training bench is much smaller than that for target application programs. In our tool, the generated training benche is simulated only about 500,000 cycles while the full simulation of the target application programs needs bilions of cycles. More detailed explanation for our method to generate the training bench is presented in [1]. The final step is to find the coefficients, c_i's corresponding to the P_i's. This is done by using multiple linear regression analysis. The energy consumption $E_{estimate}$ is then calculated using Equation (1). Figure 1 shows an overview of our energy characterization flow.

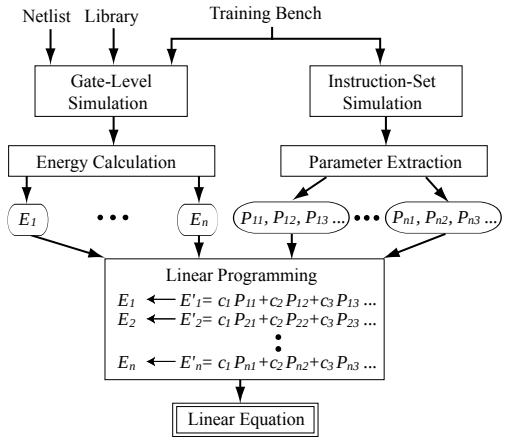

Figure 1 : Overview of Energy Characterization

To obtain the reference energy values, we simulate the processor system at gate-level for a fixed number of instructions. We refer to this number of instructions of test sequence as the instruction frame. The width is the same for all instruction frames as shown in Figure 2. Since we perform gate-level simulation and calculate the energy consumption values for all instruction frames, this step is time-consuming. However, it needs to be done only once for the characterization.

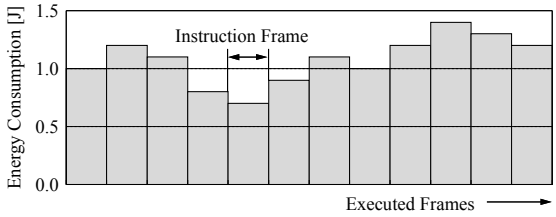

Figure 2 : An Example of Instruction Frame

We, next, obtain an instruction trace for each application program using an instruction-set simulator. The traces are divided into small segments corresponding to instruction frames. P_i's should be parameters that can be easily extracted from instruction traces. For a set of P_i's, we find coefficients which minimize $\Sigma|E_{estimate}(i)-E_{gate\text{-}level}(i)|$, where $E_{gate\text{-}level}(i)$ and $E_{estimate}(i)$ are the energy consumption values obtained by gate-level simulation and Equation (1) for the *i-th* instruction frame, respectively.

2.2 Debugger-based power estimation

Once the energy model is developed, the energy consumption of software running on the processor system can be estimated using a cycle-*inaccurate* instruction-set simulator (ISS) with the speed of 300,000 instructions per second. We use our tool for characterizing the energy consumption of two commercial microprocessors with their on-chip caches and an off-chip SDRAM. The accuracy of our method is very good as shown in Figure 3. Experimental results using three benchmark programs demonstrate that the error of our technique is on an average 3% compared to the gate-level estimation results.

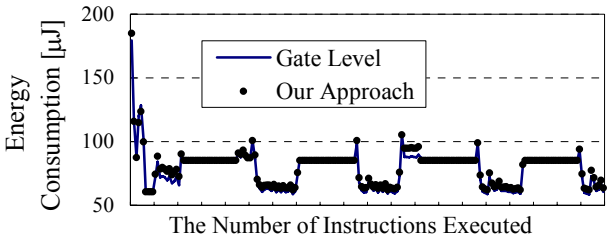

Figure 3: Results for JPEC encoder run on M32R-II

Today's SoC chips are usually implemented with off-the-shelf processor IPs. Even for those SoC chips, our method can accurately model the energy consumption since our tool does not need to know a detailed internal architecture of the target processor. Another key point of our method is that it works very well even with a cycle-*inaccurate* simulator like a GNU debugger which is a de facto standard of software debugger. This helps compilers or programmers to customize software codes to meet customers' needs for low power.

3 Memory Power Reduction

3.1 Code and data placement

This section addresses our code placement technique for reducing the total energy consumption of embedded processor systems including a CPU core, on-chip and off-chip memories. The overview of our technique for optimizing the code placement is shown in Figure 4.

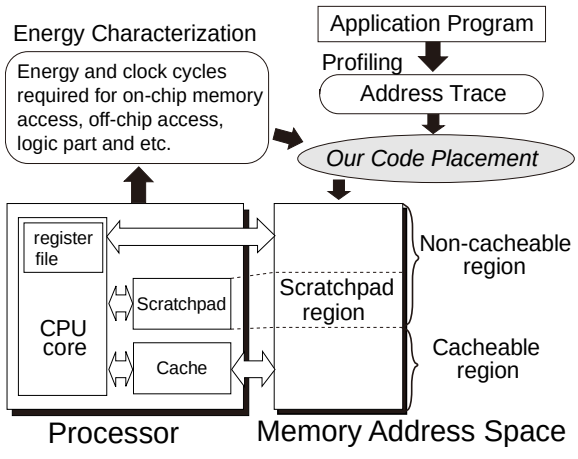

Figure 4: The Flow of Code Layout Optimization

We first extract hardware dependent parameter values like the energy consumption and clock cycles required for memory accesses using a target cell ligrary and a netlist of the target processor. Instruction and data address traces for a target application program can be obtained using instruction-set simulator (ISS). Then, our code placement algorithm simultaneously finds the optimal code allocations for a scratchpad memory, a cacheable memory region, and a non-cacheable memory region using these previously obtained hardware and software characteristics. Note that the code and data allocation is fixed at a compile time and is not changed at run time. For more detailed information of out code placement algorithm, see [5].

Figure 5 shows the energy consumption and performance results for *compress*. Left and right sides of each figure show results for 16kB and 8kB 4-way caches, respectively. 16kB, 8kB and 4kB scratchpad memories are examined as well. Vertical bar charts and straight lines represent the energy consumption and the number of cycles executed, respectively. The following five approaches are compared.

- **ORG:** Given benchmark programs are compiled with –O3 option. Every functions and data objects resides in a cacheable region in this case.
- **CHE:** Locations of functions and data objects are optimized using a conventional code placement technique. Scratchpad memory is not used in this case.
- **SPM:** Functions and data objects are relocated to a scratchpad memory using a conventional technique which maximizes the number of SPM accesses.
- **CBN:** Locations of functions and data objects are modified by applying **CHE** just after applying **SPM**.
- **OUR:** Locations of functions and data objects are optimized by our algorithm.

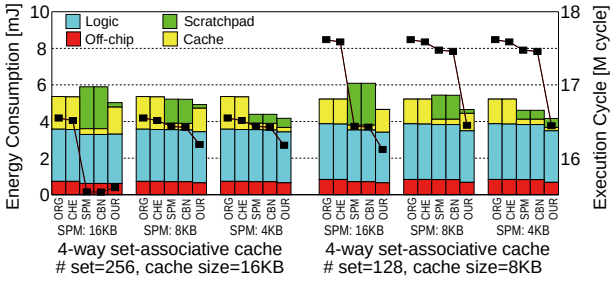

Figure 5: Optimization Results for *Compress*

Those results are obtained using a commercial embedded processor (SH3-DSP) and an off-chip SDRAM (Micron). As one can see from the results, the energy consumption of any benchmark program optimized with our approach, **OUR**, is always the smallest of all. If we employ a large scratchpad memory on a chip, the object code optimized with **SPM** or **CBN** consumes higher energy than that optimized with **CHE**. This is because the scratchpad-based compile-time memory is less energy efficient than the cache-based run-time memory if the scratchpad memory size is larger than the cache memory size. In this case, **CHE** outperforms **SPM** and **CBN** in terms of energy consumption. However, the object code optimized with **CHE** needs more execution time than that optimized with **SPM** or **CBN**. In many cases, our approach is better than the best result obtained with the other approaches in terms of both energy consumption and execution time. More specifically, for the processor with 8KB 4-way set-associative cache and 16KB scratchpad memories, our algorithm reduces the energy consumption of the processor system by 23% without any performance loss. For the 8KB 4-way set-associative cache and 4KB scratchpad memories, the result of our approach is 10% smaller in energy consumption and 6% faster in execution time compared to the best result obtained by the conventional approach. Our future work will be devoted to extend our current algorithm to find a memory configuration and the best code layout for them concurrently.

3.2 Hybrid SPM design

On-chip memories generally use higher supply (V_{DD}) and threshold (V_{th}) voltages than those of logic parts to suppress the static power consumption without increasing the access latency of the memories. This design policy, however, increases the dynamic power consumption since it is quadratically proportional to the V_{DD}. This section presents a hybrid memory architecture which consists of the following two regions; 1) a dynamic-power conscious region which uses low V_{DD} and low V_{th} and 2) a static-power conscious region which uses high V_{DD} and high V_{th}. The key of our architecture is that the access latencies for the two regions are equal to each other, which eases to integrate our memory into processors without major modifications of the internal processor architecture [3]. We can save the total power consumption by gathering the memory accesses to the dynamic-power conscious region as shown in Figure 6.

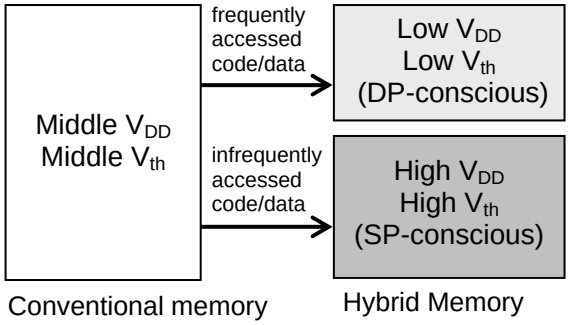

Figure 6: Code and Data Allocation for Hybrid Memory

Suppose we have two types of SRAM modules as shown in Table 1. DP and SP represent dynamic-power conscious implementation and static-power conscious implementation, respectively.

Table 1. Delay and Energy Specifications of SRAM

SRAM type	V_{DD} [V]	Process option	Delay [ps]	Leakage/ cell [nW]	Dynamic Energy [fJ]
DP	0.75	HP	197	9.27	78.1
SP	1.2	MP	198	6.71	272.0

Note that the values in Table 1 are obtained from SPICE simulation for our original SRAM modules designed with a commercial 90nm process technology. In this process technology, two process options, HP and MP, are provided. Device parameters like V_{th} and T_{ox} (i.e., gate oxide thickness) in the HP option are chosen for improving performance of the circuits. On the other hand, the paremeters in the MP option are chosen for low power design.

Once the delay and energy specifications of SRAM modules are obtained, the next step is to find code and data allocation. Our algorithm proposed in [3] finds the sizes of two regions and code/data allocations to the regions concurrently so as to minimize the total energy consumption of the hybrid scratchpad memory. Experiments using three benchmark programs, JPEG encoder, MPEG2 encoder, and compress, demonstrate that our technique reduces the energy consumption of the hybrid memory by 49% at the best case compared to the normal scratchpad design (see Figure 7).

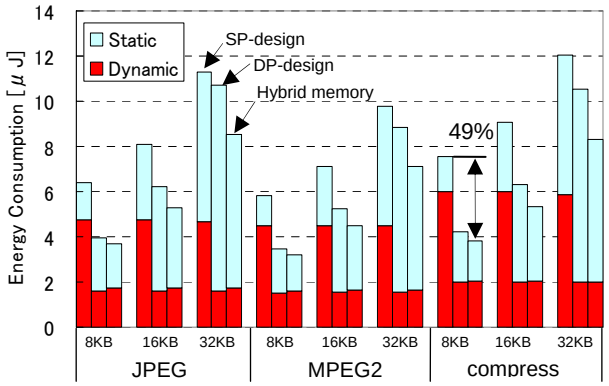

Figure 7: Power estimation results for hybrid memories

3.3 Leakage reduction by value control

Random Dopant Fluctuation (RDF) [4] within the same die results in changes in the V_{th} of transistors. Transistors of cache SRAM cells are more affected since they have minimal physical channel area. The mismatch among V_{th} of transistors of a single SRAM cell results in different leakage currents depending on the value stored in the cell. Thus leakage of the cache memory can be reduced if the values with less leakage can be more often stored in each SRAM cell. We propose techniques to reduce instruction-cache leakage using this phenomenon.

Value-Dependence of SRAM Leakage. When the SRAM cell is storing a 1 (Figure 4) only three transistors contribute to the total leakage (M1, M2, M5); when storing a 0, the other three transistors leak [5].

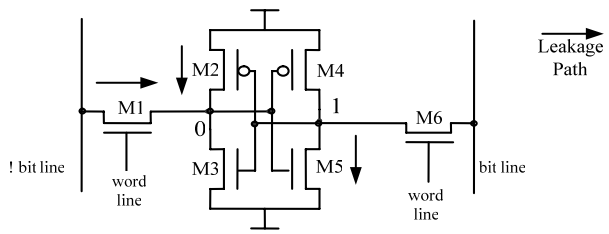

Figure 8: Different transistors leak based on cell value

Since subthreshold leakage exponentially depends on V_{th}, total leakage can be significantly different in the two states. Figure 9 shows that at 60mV variation in V_{th}, this difference is 57% averaged over cells of 1000 16KB caches. Moreover, the difference between the lowest and the highest total leakage of the cache memory is 70% on an average.

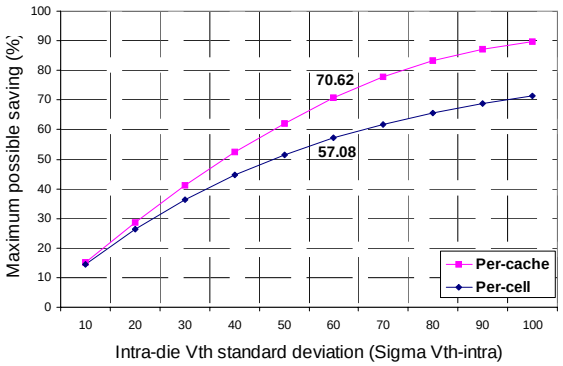

Figure 9: Leakage difference in 16KB caches.

Our Approach. We propose three techniques: *(i)* instructions within a basic-block can be rearranged (subject to dependency among them) to better match their corresponding SRAM cells in the instruction cache, *(ii)* register operands can be statically renamed to further improve the matching, *(iii)* unused lines of the cache can be initialized by their minimum-leakage value. By offline testing, the less leaky state of each SRAM cell is determined. Then, *(i)* and *(ii)* above are applied to the binary executable of each application. The third technique (cache initialization) is done at processor boot time.

Experimental Results. We applied the technique on six embedded benchmarks compiled with no compiler option for M32R processor. Figure 10 shows the leakage power breakdown to that of used and unused cache lines before and after applying our techniques when the standard deviation of within-die V_{th} variation is 60mV. The results show that the leakage power can be reduced by 54% at the best case compared to the results of a cache memory which does not use our technique. Results are averaged over 1000 simulated 8KB direct-map caches. At higher variations resulting from technology scaling, the saving increases (Figure 11). For further details see [5].

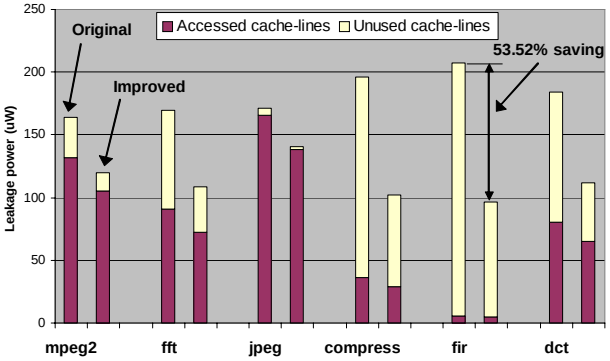

Figure 10: Average leakage on 8KB direct-map caches

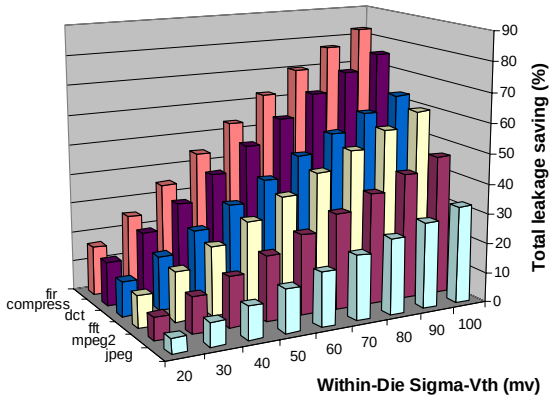

Figure 11: Saving increases with technology scaling.

3.4 Leakage reduction by way-scaling

Due to within-die variations, delays of the cells differ from one another in a cache. Thus at design time, target delay of the cache is set higher than the nominal delay to achieve an acceptable yield. Thus, if SRAM cell transistors are designed with a higher V_{th}, only some (not all) of cells will violate the target delay, but they will be randomly distributed over columns such that conventional row/column redundancy techniques are not practical. We propose to use spare cache-ways to replace the now-delay-violating cache-lines in each set (see Figure12).

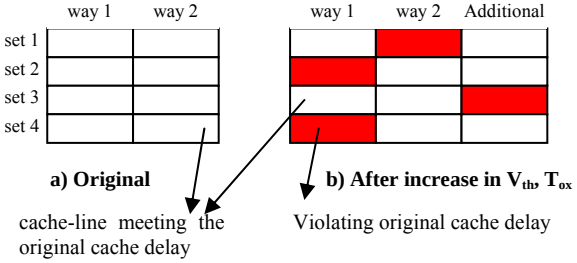

Figure 12: a) Original, b) after applying our technique

Approach. For a given target yield and target delay of a cache design, the required yield of each cell (i.e. the probability that the cell meets the target delay) can be computed. By adding each spare cache-way, this required *cell yield* is reduced, and hence, nominal *cell delay* can be increased without changing the target *cache delay* or yield. We increase *cell delay* by choosing a higher V_{th} and/or gate-oxide thickness (T_{ox}) at design time based on the number of spare ways. Additional cache-ways, however, increase dynamic energy per access, and hence, total energy does not necessarily decrease.

Experimental Results. Table 2 shows the leakage saving on a 16KB 4-way cache in a commercial 90nm process; σ/μ represents the standard deviation divided by the mean value of delay and N is the number of spare ways. Figure 13 shows total dynamic and static energy for different applications and same above L1 instruction cache. In L2 caches, where leakage is more dominant, this technique saves more energy; see Figure 14.

Table 2. Leakage reduction results using our technique

Target Yield	σ/μ	Cache leakage power (μW)			Saving (%)	
		original	N=1	N=2	N=1	N=2
90%	3%	20.552	18.030	18.773	12.27	8.66
	5%	149.80	89.123	81.191	40.51	45.80
95%	3%	21.592	18.619	19.209	13.77	11.04
	5%	166.89	95.153	84.500	42.98	49.37
99%	3%	24.501	20.014	20.171	18.32	17.67
	5%	203.29	111.83	93.502	44.99	54.00

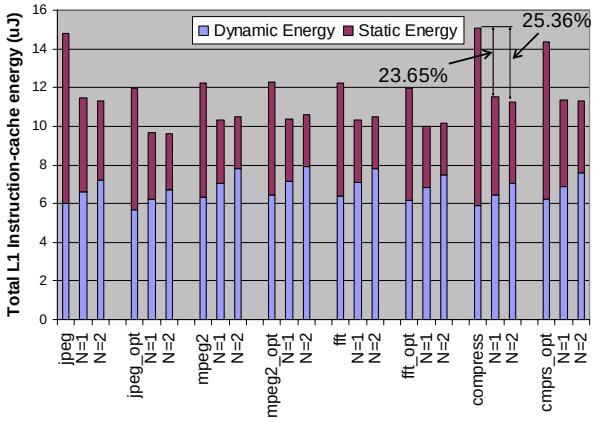

Figure 13: Results of L1 instruction-cache (16KB 4way).

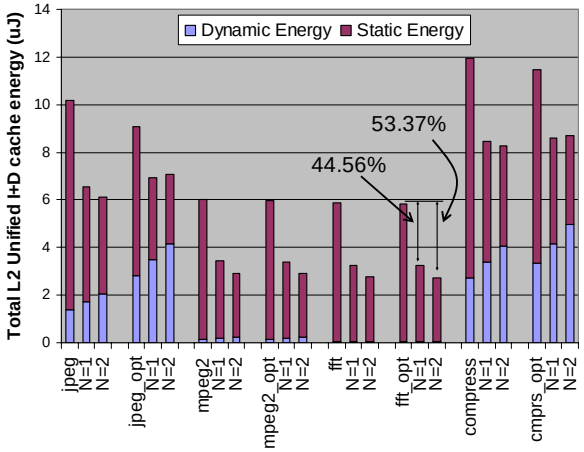

Figure 14: Results of unified L2 cache (16KB 4-way).

4 Core Logic Power Reduction

4.1 Multi-performance processor

Dynamic voltage scaling (DVS) is one of the most popular approaches for reducing the energy consumption of microprocessors. In past years, a lot of DVS processor architectures have been proposed. However, only a few of them are used in embedded real-time systems. One major reason is that most DVS processors involve large mass production cost including test cost, design cost and the cost for on-chip DC-DC converters. The other reason is a delay overhead for dynamically changing the supply voltage and the clock frequency. Reliability issue is also very serious for DVS processors in latest process technology like 65nm process. In our group, a new processor architecture which can be used as a design alternative for the DVS processors is proposed [7]. The processor core consists

of multiple same-ISA PE-cores and resizable set-associative cache memories as shown in Figure 15. Speeds and energy consumptions of PE-cores are different from each other. Only a single PE-core is selected to run at a time and the other PE-cores are deactivated by power gating and/or clock gating. The change of the active PE-core can be completed within a few clock cycles, which is suitable for the real-time applications.

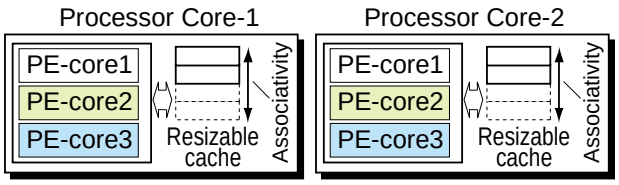

Figure 15: Muliple Performance Processor

In traditional DVS processors, a CPU core is designed to correctly work for multiple voltage conditions. In this case, critical path may be different along supply voltage even in a same chip, meaning that the DVS processor chip is not optimally synthesized for each supply voltage. Therefore, it is less power efficient than the dedicated processor core which is optimized for a specific supply voltage. Unlike the conventional DVS processors, our PE-cores are optimally designed at the pre-silicon design phase for the specific supply voltage using multiple V_{th} cells at the cost of chip area. Thus, our PE-core is more power efficient. Our processor has a 10X power scalability depending on the operating voltage and the clock frequency of the active PE-core and a cache associativity value. The power consumption of the processor can be saved by dynamically selecting the active PE-core and a cache associativity value based on the criticality of the task and the proximity of the deadline.

4.2 Synthesis results

Figure 16 shows pre-layout synthesis results of PE-cores optimized for 0.52V, 0.68V and 1.2V voltage supplies, respectively. A commercial 90nm CMOS process technology and a Media embedded Processor (MeP) of Toshiba Semiconductor are used for the experiment. Our processor can be easily synthesized using conventional synthesis flow without taking care of multiple timing constraints which should be considered in conventional DVS processor design.

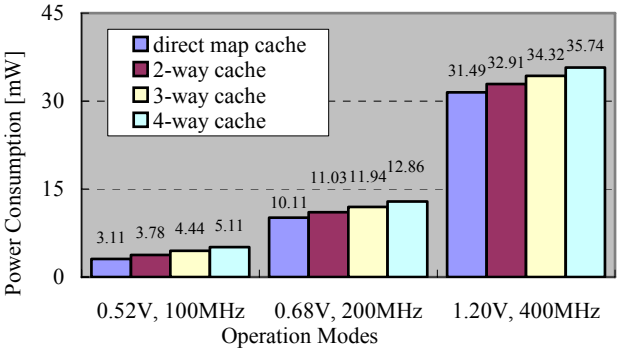

Figure 16: Pre-Layout Power Estimation Results

5 Conclusions

We presented parts of the finished as well as on-going research activities in the Circuits and Systems group of Kyushu University. Our main focus is on software-directed approaches to estimating and reducing the energy consumption of embedded real-time systems. As the demands of system integration, performance, and power have pushed ASSP vendors down to 65nm or 45nm, NRE (non-recurring engineering) costs and design complexity have increased significantly. A remedy for the NRE explosion is to reduce the number of developments and manufacture and sell tens of millions of chips under a fixed design. In such a situation, embedded software plays more and more important role than today. This paper covered our approach for fast power estimation of software on a given processor system, a number of software-directed techniques for reducing energy consumption of the memory subsystems and the logic core of the embedded processor.

Acknowledgement

This work is supported by Toshiba semiconductor and VDEC, the university of Tokyo with the collaboration of Renesas Technology, STARC, Panasonic, NEC Electronics, Toshiba, ROHM, Toppan Printing, Cadence Design Systems, Synopsys and Mentor Graphics. This work is also supported by CREST program of JST.

References

[1] D. Lee, T. Ishihara, M. Muroyama, H. Yasuura, and F. Fallah, “An Energy Characterization Framework for Software-Based Embedded Systems,” in Proc. of ESTIMedia’06, pp.59-64, Oct. 2006.

[2] Y. Ishitobi, T. Ishihara, and H. Yasuura, “Code Placement for Reducing the Energy Consumption of Embedded Processors with Scratchpad and Cache Memories,” in Proc. of ESTIMedia’07, Oct. 2007.

[3] T. Matsumura, Y. Ishitobi, M. Goudarzi, T. Ishihara, and H. Yasuura, “A Hybrid Memory Architecture for Low Power Embedded System Design,” in Proc. of SASIMI’07, Oct. 2007.

[4] Y. Taur, and T.H. Ning, Fundamentals of Modern VLSI Devices, Camrbidge University Press, 1998.

[5] M. Goudarzi, T. Ishihara, H. Yasuura, “A Software Technique to Improve Yield of Processor Chips in Presence of Ultra-Leaky SRAM Cells Caused by Process Variation,” in Proc. of ASP-DAC’07, pp. 878-883, January, 2007.

[6] M. Goudarzi, T. Ishihara, and H. Noori, “Variation-Aware Software Techniques for Cache Leakage Reduction Using Value-Dependence of SRAM Leakage due to Within-Die Process Variation," (to appear) in Proc. of HiPEAC'08, January, 2008.

[7] Y. Oyama, T. Ishihara, T. Sato, and H. Yasuura, “A Multi-Performance Processor for Low Power Embedded Applications”, in Proc. of Cool Chips X, pp. 138, April, 2007.