Exploiting Narrow Bitwidth Operations for Low Power Embedded Software Design

Yamaguchi, Seiichiro
Department of Computer Science and Communication Engineering, Graduate School of Information Science and Electrical Engineering, Kyushu University

Muroyama, Masanori
System LSI Research Center, Kyushu University

Ishihara, Tohru
System LSI Research Center, Kyushu University

Yasuura, Hiroto
Department of Computer Science and Communication Engineering, Graduate School of Information Science and Electrical Engineering, Kyushu University | System LSI Research Center, Kyushu University

出版情報：Proceedings of the Workshop on Synthesis And System Integration of Mixed Infomation Technologies, pp.51-56, 2006-04. Workshop on Synthesis And System Integration of Mixed Infomation Technologies
バージョン：
権利関係：

Exploiting Narrow Bitwidth Operations for Low Power Embedded Software Design

Seiichiro Yamaguchi† Masanori Muroyama‡ Tohru Ishihara‡ Hiroto Yasuura†‡

† Department of Computer Science and Communication Engineering,
Graduate School of Information Science and Electrical Engineering, Kyushu University
6-1 Kasuga-koen, Kasuga-shi, Fukuoka 816-8580 JAPAN
{seiichiro, yasuura}@c.csce.kyushu-u.ac.jp

‡ System LSI Research Center, Kyushu University
3-8-33 Momochihama, Sawara-ku, Fukuoka-shi, Fukuoka 814-0001 JAPAN
{muroyama, ishihara}@slrc.kyushu-u.ac.jp

Abstract - This paper proposes a low power software design technique for processor-based embedded systems. A basic idea is to reduce switching activities in sign extension bits of instruction operands through shifting the operands. To the best of our knowledge, this is the first software-level power reduction technique which exploits narrow bitwidth operations. Our technique, called *shifted operation* technique in this paper, consists of following three steps: 1) shift operands from LSB side toward MSB side by optimal shift amount, 2) execute instructions with the shifted operands, and 3) shift back computational results to original positions. Experimental results exploiting *shifted operation* technique show about 10.7% energy reduction of datapath, and about 5.5% energy reduction of overall processor.

I. Introduction

Short time to market, low cost and low power are important requirements in embedded system design. Especially, low power is the most important requirement in portable systems such as cellular phones and PDAs. In embedded system design environment, degrees of freedom in hardware are often very limited, whereas much more freedom is available in software. In addition to the benefit, software-oriented optimization is applicable to general purpose processors. In this paper, we propose a low power software design technique for processor-based embedded systems.

Processors are implemented by digital CMOS circuits. There are three major sources of power consumption in digital CMOS circuits. These sources are switching power, short-circuit power and leakage power [1]. In particular, the switching power for charging and discharging of load capacitance is the most major source of the power consumption. The switching power is shown as follows:

$$P_{sw} = \alpha \cdot C_L \cdot V_{dd}^2 \cdot f_{clk} \quad \cdots(1)$$

where α denotes the switching activity factor, C_L represents the load capacitance, V_{dd} is the supply voltage and f_{clk} is the clock frequency. According to (1), several approaches can reduce the switching power. In this paper, we focus on reducing the switching activity factor. Since the short-circuit power is consumed at the time of transistor switching, the short-circuit power can be reduced through reducing the switching activity factor as well. The switching power and the short-circuit power are called dynamic power, which is our target to reduce.

To design a low power processor-based embedded system, we propose a novel low power design technique exploiting narrow bitwidth operations at software-level. This is the first software-level power reduction technique. Since software-level power reduction techniques do not require any hardware modifications, the techniques can be easily applied to existing processor-based embedded systems. Therefore, our technique satisfies the requirements of short time to market and low cost. Two's complement representation is typically chosen to represent numbers since arithmetic operations are easy to perform in processor systems. One of problems with two's complement representation is sign extension. Due to sign extension, an arithmetic operation for narrow bitwidth operands requires the dynamic power throughout entire word length. We reduce the dynamic power consumption due to sign extension through shifting the narrow bitwidth operands. Our technique, called *shifted operation* technique in this paper, consists of following three steps: 1) shift operands from LSB side toward MSB side by optimal shift amount, 2) execute instructions with the shifted operands, and 3) shift back computational results to original positions. *Shifted operation* technique is effective to overall datapath of processor, because sign extension affects the dynamic power consumption of not only arithmetic circuits, but also buses, registers and logical circuits.

In this paper, we present a mechanism of the power reduction exploiting narrow bitwidth operations. To find optimal shift amount for each variable of a target application source code, we formulate *shift amount optimization problem*. *Shifted operation* technique generates low energy source codes through inserting shift operations to the target application source code according to the optimal shift amount for each variable. This paper is organized as follows: Section II discusses related work and presents our approach. In Section III, we formulate *shift amount optimization problem*. Experimental results are presented in Section IV. Finally, Section V concludes this paper.

II. Related Work and Our Approach

A. Previous Work

There are several approaches for reducing the switching activities in sign extension bits. Using other number representation is one of the approaches. *Sign-magnitude* representation in which only one bit is allocated for the sign and the rest for the magnitude [2]. *Significance compression* is also effective through appending two or three extra bits to represent significant bytes [3]. *Reduced two's-complement*

representation generates a representation dynamically according to a magnitude of number [4]. In addition, *low power adder operating* which considers narrow bitwidth operands dynamically is proposed in [5], and several bus coding techniques are discussed in [6], [7], [8]. Brooks et al. showed that over half of integer operation executions require 16-bits or less across SPECint95 benchmarks [9]. We also exploit this fact for low power embedded software design.

These techniques mentioned above need some hardware modifications. On the other hand, *shifted operation* technique does not need any hardware modifications. Therefore, *shifted operation* technique can be applied to existing processor-based embedded systems.

B. Motivational Example

To illustrate a key point of *shifted operation* technique, we introduce a motivational example by using 32-bits Ripple Carry Adder (RCA). In Fig. 1, two input operands, *x(t)* and *y(t)*, represented by two's complement are added where *x(t-1)* and *y(t-1)* are previous input operands. Since the dynamic power of RCA originates from the switching activities of two input operands changing from *x(t-1)* and *y(t-1)* to *x(t)* and *y(t)*, four input operands have to be considered. Significant bits of all the operands are only 8-bits. Rectangles indicate significant bits of the operands. The MSB in significant bits is the sign bit. Remaining upper 24-bits are sign extension bits for conventional operation. Meanwhile, lower 24-bits are all '0' for *shifted operation* because the operands are shifted from LSB side toward MSB side firstly. A large number of the switching activities due to sign extension are generated when conventional operation is executed. On the other hand, there is no switching activity due to sign extension when *shifted operation* is applied.

We compared the dynamic power consumptions at gate-level by using Cadence NC-Verilog to count the switching activities of all nets, and using Synopsys Power Compiler to estimate the dynamic power consumptions. A process technology we used is Hitachi CMOS 0.18μm standard cell library. Supply voltage is 2.5V and clock frequency to RCA is 10MHz. In this case, the dynamic power consumptions of RCA are 61.3μW for conventional operation and 12.0μW for *shifted operation*. The dynamic power consumption is reduced 80.4% through applying *shift operation* technique. Note that we ignored overheads to shift operands. Of course, we consider the overheads and discuss their reduction in the following subsection.

C. Our Approach

This subsection presents an overview of our approach. Previous subsection introduced the power reduction mechanism of *shifted operation* technique. However, there is a huge issue on the overheads due to shift operations. There is no guarantee that narrow bitwidth operations are always executed on datapath. In other words, not all operands can be shifted. Another point to notice is that operational bitwidth for each instruction may be difference each other. We cannot insert shift operations for each instruction because of the

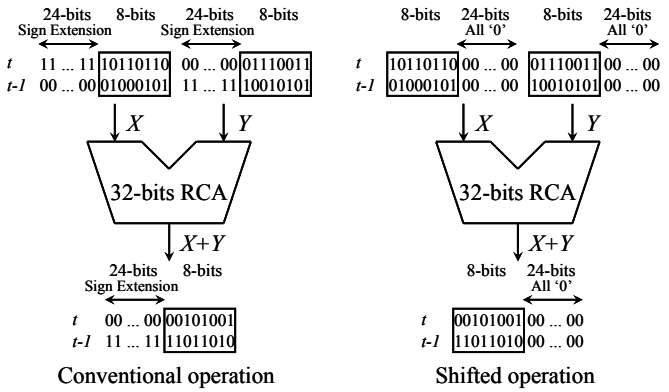

Fig. 1. Motivational Example by using 32-bits Ripple Carry Adder.

overheads. It is clear that the overheads dramatically increase if we insert shift operations before and after every instruction. To keep down the overheads, we shift each variable of a target application source code only once according to the operational bitwidth. Since the dynamic power consumption strongly depends on the operational bitwidth and the overheads, it is necessary to find optimal shift amount for each variable. We should consider additional time/power overheads due to shift operations. *Shifted operation* technique reduces energy consumption which is a product of average power consumption and execution time. Fig. 2 shows the overview of our approach, and details of the overview are as follows:

Given data [abbr.]:
- C source code of target application [Target-C]
- Chip or HDL code of target processor [IP]
- Target process technology [Process]
- Training data for instruction trace [Data1]
- Training data for energy characterization [Data2]

Given tools [abbr.]
- Compiler for the target processor [Compiler]
- Instruction set simulator for the target processor [ISS]
- Energy estimation tools [Tool]

Procedure prototype [abbr.] (in1, ...) - out1 [abbr.], ...:
- Compile [Compile] (Target-C, Compiler)
  - Assembler source code of the target application [Asm]
  - Object code of the target application [Obj]
- Variable size analysis [VSA] (Target-C)
  - Effective bitwidth for each variable [EB]
- Operand size analysis [OSA] (Asm, EB)
  - Minimum operational bitwidth for each instruction [MOB]
- Instruction trace [Trace] (Obj, Data1, ISS)
  - Trace data [T-Data]
- Energy characterization [EC] (IP, Process, Data2, Tool)
  - Energy consumption model for each circuit module [EM]
- Shift amount analysis for each variable [SAA] (EB, MOB, T-Data, EM)
  - Optimal shift amount for each variable [OSA]
- Code Optimization [Optimization] (Target-C/Asm, OSA)
  - Optimized C/assembler source code [Optimized-C/-Asm]

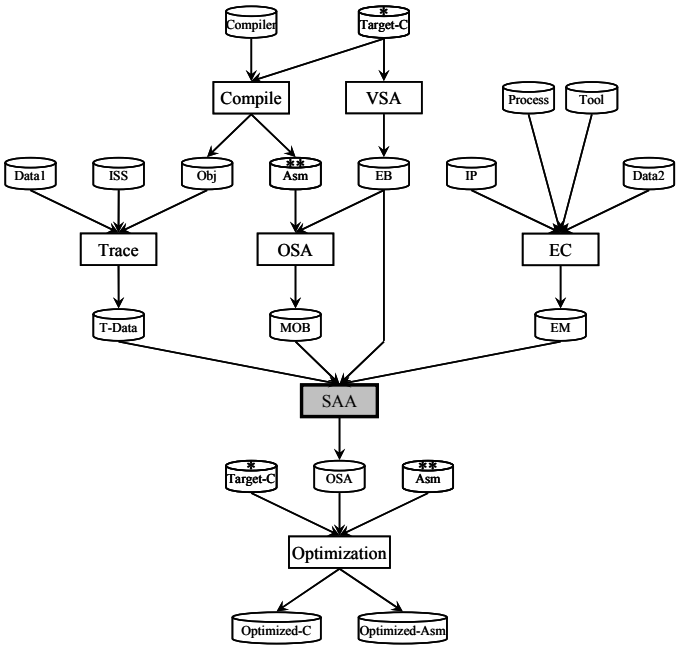

Fig. 2. Overview of our approach to generate a low energy C/assembler source code.

Output [abbr.]:

- Optimized C/assembler source code [Optimized-C/-Asm]

Four parameters are needed to find the optimal shift amount for each variable: effective bitwidth for each variable, minimum operational bitwidth for each instruction, trace data and energy consumption model for each circuit module.

First of all, we compile C source code of a target application, and find the effective bitwidth for each variable by variable size analysis. After that, the minimum operational bitwidth for each instruction is found by operand size analysis. We define the effective bitwidth as a smallest bitwidth which can hold both maximum and minimum values of variable or operand, and the minimum operational bitwidth as the largest effective bitwidth of instruction operands. In many cases, some bits of variables are never used during execution of the target application program. For example, if an integer type variable *x* of which value is in [-500, 500], between -500 and 500, then the effective bitwidth of *x* is 10-bits, because 10-bits are enough to hold any value in [-500, 500]. There are two approaches to analyze the effective bitwidth [10]. One is static analysis, and the other is simulation-based dynamic analysis. In *shifted operation* technique, variable size analysis and operand size analysis apply the static analysis. Next, to obtain the trace data, we simulate the object code with the training data by using instruction set simulator. Energy characterization estimates the average dynamic power consumptions for each circuit modules by using power estimation tools. Then, we calculate products of the average dynamic power consumptions and the execution times as the energy consumption model for each circuit module.

After obtaining these four parameters, we find the optimal shift amount for each variable by shift amount analysis. Shift amount analysis, which is the main procedure of *shifted operation* technique, is to solve *shift amount optimization problem* formulated in the next section. Finally, low energy C/assembler source codes are generated through inserting shift operations to original C/assembler source code according to the optimal shift amount for each variable.

III. Problem Description

A. Notation

We define notations used in formulation of *shift amount optimization problem*.

- M: Number of target circuit modules, such as adders, multipliers, logic units, latches and buses.
- N_m: Number of instructions which appeared in the object code and can be executed on circuit module m. Instruction is identified by its memory address.
- $P_{m,i}$: Number of instructions which can be executed on circuit module m immediately before instruction i.
- W: Datapath width of the target processor.
- $E_{m,i,i',w,w'}$: Energy consumption model of circuit module m when operational bitwidth of instruction i is w-bits and operational bitwidth of instruction i' is w'-bits immediately before instruction i. Note that the $E_{m,i,i',w,w'}$ can be modeled by following parameters:
  - Type of circuit module m on which instruction i and i' are executed.
  - Operational bitwidth of instruction i (w-bits).
  - Operational bitwidth of instruction i' (w'-bits).
  - Effective bitwidths for each operand of instruction i.
  - Effective bitwidths for each operand of instruction i'.
- $X_{m,i,i'}$: Number of executions for instruction i executed on circuit module m immediately after instruction i'.
- B_i: Minimum operational bitwidth which guarantees instruction i to be executed without sacrificing any computational precisions.
- $v_{m,i,i',w,w'}$: 0-1 integer variable to be determined. The variable is set to 1 if instruction i is executed on circuit module m with w-bits and instruction i' is executed with w'-bits immediately before instruction i. Otherwise it is set to 0.

B. ILP Formulation

Shift amount optimization problem is formulated as follows:

Objective function to be minimized:

$$\sum_{m=1}^{M}\sum_{i=1}^{N_m}\sum_{i'=1}^{P_{m,i}}\sum_{w=1}^{W}\sum_{w'=1}^{W} E_{m,i,i',w,w'} \cdot X_{m,i,i'} \cdot v_{m,i,i',w,w'} \quad \cdots (2)$$

Subject to:

For each i and i' $$\sum_{w=1}^{W}\sum_{w'=1}^{W} v_{m,i,i',w,w'} = 1 \quad \cdots(3)$$

where i' denotes an instruction executed on circuit module m immediately before instruction i. Note that a number of i's can be more than one.

For each i and i' $$\sum_{w=1}^{W}\sum_{w'=1}^{W} w \cdot v_{m,i,i',w,w'} \geq B_i \cdots(4)$$

where i' denotes an instruction executed on circuit module m immediately before instruction i. Note that a number of i's can be more than one.

For each i and i' $$\sum_{w=1}^{W}\sum_{w'=1}^{W} w' \cdot v_{m,i,i',w,w'} = \sum_{w'=1}^{W}\sum_{w''=1}^{W} w' \cdot v_{m,i',i'',w',w''} \cdots(5)$$

where i' denotes an instruction executed on circuit module m immediately before instruction i. Note that a number of i's can be more than one.

For each i and j $$\sum_{w=1}^{W}\sum_{w'=1}^{W} w \cdot v_{m,i,i',w,w'} = \sum_{z=1}^{W}\sum_{z'=1}^{W} z \cdot v_{m,j,j',z,z'} \cdots(6)$$

where j denotes an instruction which shares at least one variable with instruction i. Note that a number of js can be more than one.

The first constraint (3) indicates that only one operational bitwidth should be assigned to each instruction. The second constraint (4) means that operational bitwidth for instruction i must be greater than or equal to the minimum operational bitwidth which guarantees instruction i to be executed without sacrificing any computational precisions. The third constraint (5) indicates that operational bitwidth w' determined by $v_{m,i,i',w,w'}$ and that determined by $v_{m,i',i'',w',w''}$ should be consistent. The last one (6) means that operational bitwidths of instructions which use a same variable as its operand should be equal to each other. *Shift amount optimization problem* can be formally defined as follows:

"For given $P_{m,i}$, $E_{m,i,i',w,w'}$, $X_{m,i,i'}$ and B_i, find a set of $v_{m,i,i',w,w'}$ which minimizes the objective function (2) under constraints of (3), (4), (5) and (6)."

Clearly *shift amount optimization problem* defined above is an integer linear programming (ILP) problem. Therefore, complexity of the problem depends on a polynomial of the number of variables to be determined ($M \times N^2 \times W^2$) and the number of constraints ($M \times N^2$). Here N denotes the number of instructions in the object code. *Shift amount optimization problem* cannot be solved within a practical time if M, N and W are large numbers. Our idea for reducing the complexity of the problem is to reduce N and W. Operational bitwidth for each instruction can be any integer values between 0 to W. However, if we limit the shift amount into predefined values, the complexity can be reduced. For example, if we limit the shift amount into multiples of 4, the problem size can be quartered. Since N is the number of instructions in the object code, N can be decreased by reducing the target code size. For example, we can reduce N by applying our technique to frequently executed functions only. If the complexity is still high, then we need to come up with new heuristic algorithm.

IV. Experiments and Results

A. Experimental Framework

To evaluate the effectiveness of *shifted operation* technique, we experiment under following conditions:

Given data:
- Target-C: C source code of 16-points moving average filter
- IP: HDL code of M32R-II processor core (32-bits RISC processor of Renesas Technology)
- Process: Hitachi CMOS 0.18μm standard cell library
- Data1: Sign wave (# of samples: 4096, 1 sample: 16-bits)
- Data2: Random data

Given tools:
- Compiler: m32r-linux-gcc
- ISS: m32r-linux-run
- Tool: Cadence NC-Verilog, Synopsys Design Compiler and Synopsys Power Compiler

Other conditions, supply voltage is 2.5V and clock frequency to M32R-II processor core is 10MHz. Fig. 3 shows the architecture of 16-points moving average filter we used as Target-C.

We use an optimization option "-O3" when we compile the C source codes. Energy characterization estimates the average dynamic power consumptions for each circuit module with synthesized HDL code of processor core by using Synopsys Design Compiler to synthesize the HDL code, and using Cadence NC-Verilog to count the switching activities of all net, and using Synopsys Power Compiler to estimate the average power consumptions. We insert shift operations to original source codes by hand according to the results of *shift amount optimization problem* which is the optimal shift amount for each variable.

After inserting shift operations, we compile the optimized C source code with an optimization option "-O3", and assemble the optimized assembler source code with no options. In the first case, there is a possibility that the original object code and the optimized one are different. In the latter case, those object codes are always same except shift operations inserted by *shifted operation* technique. This means that the *shifted operation* technique at assembler source-level has execution cycle overheads absolutely. These three object codes are simulated on the synthesized HDL code of processor core to estimate the average dynamic power consumptions, and to obtain the execution times.

B. Experimental Results

The result of variable size analysis for 16-points moving average filter is tabulated in TABLE I. Since the bitwidth of

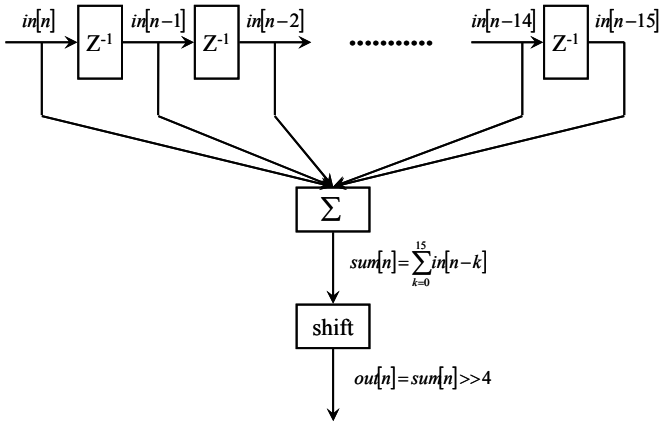

Fig. 3. Architecture of 16-points moving average filter.

TABLE I
Result of Variable Size Analysis for 16-points Moving Average Filter

Variable Name	Effective Bitwidth	Variable Name	Effective Bitwidth
in[n]	16	*in[n-9]*	16
in[n-1]	16	*in[n-10]*	16
in[n-2]	16	*in[n-11]*	16
in[n-3]	16	*in[n-12]*	16
in[n-4]	16	*in[n-13]*	16
in[n-5]	16	*in[n-14]*	16
in[n-6]	16	*in[n-15]*	16
in[n-7]	16	*sum[n]*	20
in[n-8]	16	*out[n]*	16

input data is 16-bits, the effective bitwidth of all the variables except *sum[n]* are 16-bits. However, we cannot shift the variables by 16-bits, because the minimum operational bitwidth is defined as the largest effective bitwidth of instruction operands.

Fig. 4 shows power estimation results of 32-bits RCA for *shifted operation* technique while changing the operational bitwidth at *t* and *t-1*, respectively. Assume that the effective bitwidths of input operands are equal to each other, and equal to the operational bitwidth. Therefore, if the operational bitwidth is 8-bits, then lower 24-bits of the operands are all '0'. Fig. 3 shows that the average dynamic power consumption is roughly proportional to the larger operational bitwidth either at *t* or *t-1*.

We find that the optimal shift amount for all the operands are 12-bits through solving *shift amount optimization problem*. Additional time overheads due to shift operations are shown in TABLE II. The number of execution cycles increases only 1.2% in assembler source-level optimization. On the other hand, the number of execution cycles decrease 3.4% in C source-level optimization. We have to discuss about reasons of the results. This is our future work.

TABLE III shows both cases can reduce power/energy consumption of datapath even if assembler source-level optimization increased the number of execution cycles. The results show about 7.6%/4.9% power reduction and about 10.7%/3.7% energy reduction for *shifted operation* technique at C/assembler source-level optimization. In addition to the

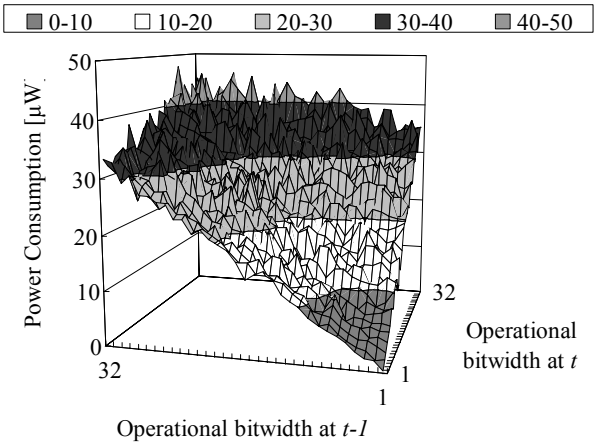

Fig. 4. Power estimation results of 32-bits RCA for *shifted operation* technique.

TABLE II
Number of Execution Cycles and Instructions

Object Code	Execution Cycles	Instructions
Original	289,246	204,844
Optimized-C	279,520 (-3.4%)	204,843 (0.0%)
Optimized-Asm	292,833 (1.2%)	208,940 (2.0%)

TABLE III
Power/Energy Estimation Results of Datapath

Object Code	Module	Power [mW]	Energy [mJ]
Original	Registers	12.09	349.7
	Buses	4.73	136.8
	ALU	4.71	136.2
	Shifter	0.52	15.1
	Total	22.05	637.9
Optimized-C	Registers	11.43 (-5.5%)	319.4 (-8.7%)
	Buses	4.17 (-11.9%)	116.5 (-14.8%)
	ALU	4.01 (-14.9%)	111.9 (-17.8%)
	Shifter	0.78 (48.6%)	21.7 (43.6%)
	Total	20.39 (-7.6%)	569.6 (-10.7%)
Optimized-Asm	Registers	11.49 (-5.0%)	336.5 (-3.8%)
	Buses	4.51 (-4.8%)	132.0 (-3.6%)
	ALU	4.21 (-10.6%)	123.3 (-9.5%)
	Shifter	0.76 (45.9%)	22.3 (47.7%)
	Total	20.97 (-4.9%)	614.1 (-3.7%)

TABLE IV
Power/Energy Estimation Results of Overall Processor Core

Object Code	Power [mW]	Energy [J]
Original	76.65	2.22
Optimized-C	74.99 (-2.2%)	2.10 (-5.5%)
Optimized-Asm	74.56 (-2.7%)	2.18 (-1.5%)

effectiveness, TABLE IV shows about 2.2%/2.7% power reduction and about 5.5%/1.5% energy reduction of overall processor.

V. Conclusions

We have proposed a novel low power software design technique, called *shifted operation* technique, exploiting narrow bitwidth operations for processor-based embedded systems. *Shifted operation* technique consists of following three steps: 1) shift operands from LSB side toward MSB side by optimal shift amount, 2) execute instructions with the shifted operands, and 3) shift back computational results to original positions. We also have presented *shift amount optimization problem* as a 0-1 integer linear programming problem. Experimental results exploiting *shifted operation* technique have shown about 10.7% energy reduction of datapath and about 5.5% energy reduction of overall processor. Our future work are as follows:

- To define an algorithm to solve *shift amount optimization problem*
- To discuss reasons of the energy reduction
- To experiment by using other benchmarks and other processors

Acknowledgements

This work has been supported in part by the Grant-in-Aid for Creative Scientific Research No. 14GS0218 and the grant of Fukuoka project in the Cooperative Link of Unique Science and Technology for Economy Revitalization (CLUSTER) of the Ministry of Education, Culture, Sports, Science and Technology (MEXT). This work is supported by Core Research for Evolutional Science and Technology (CREST) of Japan Science and Technology Corporation (JST). This work is supported by VLSI Design and Education Center (VDEC), The University of Tokyo with the collaboration of Renesas Technology, Hitachi, Ltd, Cadence Design Systems, Inc. and Synopsys, Inc. We are grateful for their support.

References

[1] N. H. E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design - A System Perspective - Second Edition*, Addison-Wesley, 1993.

[2] A. P. Chandrakasan and R. W. Brodersen, “Minimizing Power Consumption in Digital CMOS Circuits,” *in Proceedings of the IEEE*, vol. 83, no. 4, pp.498-523, April 1995.

[3] R. Canal, A. González and J. E. Smith, “Very Low Power Pipelines using Significance Compression,” *in Proceedings of the 33rd International Symposium on Microarchitecture*, pp.181-190, 2000.

[4] Z. Yu, M.-L. Yu, K. Azadet and A. N. Willson., Jr. “The Use of Reduced Two’s-Complement Representation in Low-Power DSP Design,” *in Proceedings of IEEE International Symposium on Circuit and Systems*, vol. 1, pp.I-77-I-80, May 2002.

[5] O. T.-C. Chen, R. R.-B. Sheen and S. Wang, “A Low-Power Adder Operating on Effective Dynamic Data Ranges,” *IEEE Transactions on Very Large Scale Integration Systems*, vol. 10, no. 4, August 2002.

[6] M. R. Stan and W. P. Burleson, “Bus-Invert Coding for Low Power I/O,” *IEEE Transactions on Very Large Scale Integration Systems*, vol. 3, no. 1, pp.49-58, March 1995.

[7] M. Muroyama, A. Hyodo, T. Okuma and H. Yasuura, “A Power Reduction Scheme for Data Buses by Dynamic Detection,” *IEICE Transactions on Electronics*, vol. E87-C, no. 4, April 2004.

[8] M. Saneei, A. A.-Kusha and Z. Navabi, “Sign Bit Reduction Encoding for Low Power Applications,” *in Proceedings of the 42nd Design Automation Conference*, pp.214-217, June 2005.

[9] D. Brooks and M. Martonosi, “Dynamically Exploiting Narrow Width Operands to Improve Processor Power and Performance,” *in Proceedings of the 5th International Symposium on High Performance Computer Architecture*, pp.13-22, January 1999.

[10] H. Yamashita, H. Yasuura, F. N. Eko, and Y. Cao, “Variable Size Analysis and Validation of Computation Quality,” *in Proceedings of High-Level Design Validation and Test Workshop*, pp.95-100, November 2000.