Exploiting Narrow Bitwidth Operations for Low Power Embedded Software Optimization

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

Exploiting Narrow Bitwidth Operations for Low Power Embedded Software Optimization

Seiichiro Yamaguchi†, Muroyama Masanori‡, Tohru Ishihara‡ and 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 814-0001 Japan
{muroyama, ishihara}@slrc.kyushu-u.ac.jp

Abstract - This paper proposes a low power software optimization 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. Our technique, called *shift operation insertion* 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 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-level optimization is applicable to general purpose processors. In this paper, we propose a low power software optimization 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 power reduction technique exploiting narrow bitwidth operations at software-level. This is the first software-level power reduction technique which exploits narrow bitwidth operations. Since software-level power reduction techniques do not require any hardware modifications, the techniques can be easily applied to existing processor-based embedded systems. 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 *shift operation insertion* 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. *Shift operation insertion* technique is effective to 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 *optimal shift amount finding problem*. *Shift operation insertion* technique generates low power 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 *optimal shift amount finding 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]. Brooks et al. showed that over half of integer operation executions require 16-bits or less across SPECint95 benchmarks [8]. We also exploit this fact for *shift operation insertion* technique.

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

B. Motivational Example

To illustrate a key point of *shift operation insertion* 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. On the other hand, lower 24-bits are all '0' for *shift operation insertion* technique 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 *shift operation insertion* technique 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 *shift operation insertion* technique. The dynamic power consumption is reduced 80.4% through applying *shift operation insertion* 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

Previous subsection introduced the motivational example of *shift operation insertion* technique. However, there is a huge issue on the overheads due to shift operations. There is no guarantee that narrow bitwidth

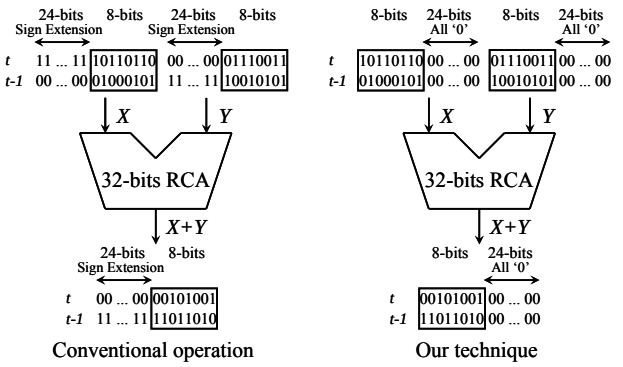

Fig. 1: Motivational example by using 32-bits RCA.

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 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 input/output variables of a basic block 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. *Shift operation insertion* technique reduces energy consumption which is a product of average power consumption and execution time.

Three parameters are needed to find the optimal shift amount for each variable: effective bitwidth for each variable, minimum operational bitwidth for each instruction and energy consumption model for each circuit module. We define the effective bitwidth as a smallest bitwidth which can hold both maximum and minimum values. 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 [9]. One is static analysis, and the other is simulation-based dynamic analysis. In *shift operation insertion* technique, we use the static analysis. The minimum operational bitwidth of an instruction is defined as the largest effective bitwidth of input/output operands. Fig. 2 shows power estimation results of 32-bits RCA for *shift operation insertion* 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, lower 24-bits of the operands are all '0'. Fig. 2 shows that power consumptions of 32-bits RCA are roughly proportional to larger operational bitwidth either at *t* or *t-1*. In this paper, we assume that all of energy consumptions for each circuit module are proportional to larger operational bitwidth.

After obtaining these four parameters, we find the optimal shift amount for each variable by solving *optimal shift amount finding problem* formulated in the next

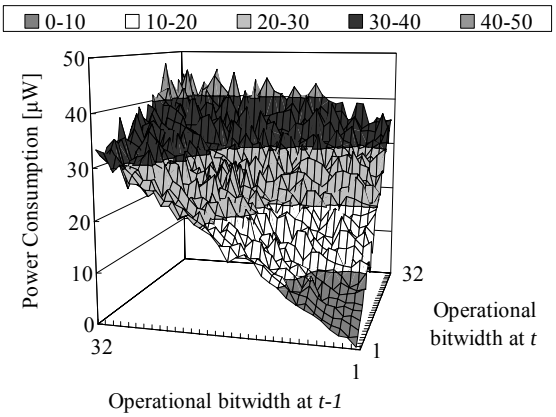

Fig. 2: Power estimation results of 32-bits RCA for *shift operation insertion* technique.

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 *optimal shift amount finding problem*.

- N_M: Number of circuit modules existing on datapath, such as adders, multipliers, logic units, latches and buses.
- N_I: Number of instructions which appeared in a target basic block. Instruction is identified by its memory address.
- B_i: Minimum operational bitwidth of instruction i.
- W: Datapath width of a 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.
- $v_{m,i,i',w,w'}$: 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. Formulation

Optimal shift amount finding problem is formulated as follows:

Minimize:

$$\sum_{m=1}^{N_M}\sum_{i'=0}^{N_I-1}\sum_{i=i'+1}^{N_I}\sum_{w=B_i}^{W}\sum_{w'=B_{i'}}^{W} E_{m,i,i',w,w'}\cdot v_{m,i,i',w,w'} \quad \cdots(2)$$

Subject to:

For each i

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

For each i

$$\sum_{w=B_i}^{W}\sum_{w'=B_{i'}}^{W} w'\cdot v_{m,i,i',w,w'} = \sum_{w'=B_{i'}}^{W}\sum_{w''=B_{i''}}^{W} w'\cdot v_{m,i',i'',w',w''} \cdots(4)$$

For each i and j

$$\sum_{w=B_i}^{W}\sum_{w'=B_{i'}}^{W} w\cdot v_{m,i,i',w,w'} = \sum_{z=B_j}^{W}\sum_{z'=B_{j'}}^{W} z\cdot v_{m,j,j',z,z'} \cdots(5)$$

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.

Find:

a set of $v_{m,i,i',w,w'}$

The first constraint (3) indicates that only one operational bitwidth should be assigned to each instruction. The second constraint (4) means 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 (5) indicates that operational bitwidths of instructions which use a same variable as its operand should be equal to each other. *Optimal shift amount finding problem* can be formally defined as follows:

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

Clearly *optimal shift amount finding problem* defined above is a 0-1 integer programming problem.

IV. EXPERIMENTS AND RESULTS

A. Experimental Framework

To evaluate the effectiveness of *shift operation insertion* technique, we experimented under following conditions:

- Target application: 16-points moving average filter
- Sample input: Sign wave (short type integer)
- Target processor: M32R-II processor core (32-bits RISC processor of Renesas Technology)
- Clock frequency: 10MHz
- Process technology: Hitachi CMOS 0.18μm
- Supply voltage: 2.5V

Fig. 3 shows the architecture of 16-points moving average filter. Input variable is *in*[*n*] and output variable is *out*[*n*]. We used an optimization option "-O3" when we compiled the original/optimized C source codes. On the other hand, we assembled an 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. This means that *shift operation insertion* technique at assembler source-level has execution cycle overheads absolutely.

B. Experimental Results

Since the input dataset is short type integer, the effective bitwidths of *in*[*n*]-*in*[*n-15*] and *out*[*n*] are 16-bits, and *sum*[*n*] is 20-bits. We found the optimal shift amount for all the variables are 12-bits by solving *optimal shift amount finding problem*. Additional time overheads due

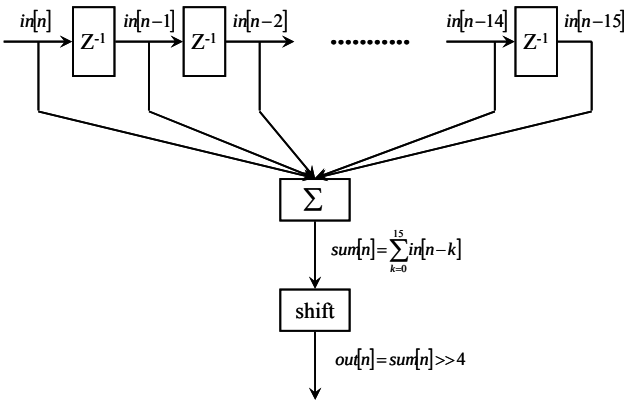

Fig. 3: Architecture of 16-points moving average filter.

to shift operations are summarized in TABLE I. The number of execution cycles increases only 1.2% in assembler source-level optimization. On the other hand, it decreases 3.4% in C source-level optimization. TABLE II 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 *shift operation insertion* technique at C/assembler source-level optimization. In addition to the effectiveness, about 2.2%/2.7% power reduction and about 5.5%/1.5% energy reduction of overall processor for *shift operation insertion* technique at C/assembler source-level optimization.

TABLE I: Number of Execution Cycles and Instructions

Object code	Instructions	Execution cycles
original	204,844	289,246
optimized C	204,843 (-0.0%)	279,520 (-3.4%)
optimized Asm	208,940 (2.0%)	292,833 (1.2%)

TABLE II: 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 (46.3%)
	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%)	123.0 (-9.5%)
	Shifter	0.76 (45.9%)	22.3 (47.7%)
	Total	20.97 (-4.9%)	614.1 (-3.7%)

V. CONCLUSIONS

We have proposed a novel low power software optimization technique, called *shift operation insertion* technique in this paper, exploiting narrow bitwidth operations for processor-based embedded systems. *Shift operation insertion* 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 *optimal shift amount finding problem* as a 0-1 integer programming problem. Experimental results exploiting *shift operation insertion* technique have shown about 10.7% energy reduction of datapath and about 5.5% energy reduction of overall processor. Our future work are to discuss detailed reasons of the energy reduction and 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, December 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. 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.

[7] 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.

[8] 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.

[9] 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.