社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2003－ＳＬＤＭ－110　(9)
2003／5／16

Dynamic Access Control of Instruction Buses by Using Active Bits Detection for Low Power Systems

Masanori Muroyama　Akihiko Hyodo　Takanori Okuma　Koji Makiyama and　Hiroto Yasuura

Department of Computer Science and Communication Engineering, Kyushu University
6-1 Kasuga Koen, Kasuga-shi, Fukuoka, 816-8580 Japan

E-mail:　{muroyama, akihiko, okuma, makiyama, yasuura}@c.csce.kyushu-u.ac.jp

Abstract　External buses consume substantial power for their high capacitances of bus lines and I/O pins. In this paper, we reduce redundant dynamic power consumption on instruction buses. For expressing a small number, we need the small number of bits (the minimum bits are defined as active bits). But all bit lines on an instruction bus change their status and redundant power is consumed. Suppressing switching activity of inactive bits, we can reduce redundant power consumption. Since the active bits are different each data, we reduce power by dynamic analysis and control of the instruction buses. An instruction is constructed of several fields which indicate kinds of operations, register numbers and an immediate value. After analyzing the characteristics of active bits in each field by profiling we change opcode and register assignments by using variable length coding to reduce active bits. We achieve more efficient power reduction by using active bits with the code assignment of instruction sets. Experimental results illustrate up to 29.1% switching activity reduction for MPEG2.

Keyword　Low-Power, Instruction Bus, Active Bits, Dynamic Power.

1. Introduction

Nowadays, to decrease power consumption of LSI is strongly desired from demands of increasing battery life and the problem of energy resources. Reducing power consumption realizes high reliability and low cost for cooling and other advantages. Under present situation we must resign high performance in some degree to restrain power consumption. In particular microprocessors which occupy important position in a system consume large power in the system. Many microprocessors are embedded in computers and home electric appliances and cars and so on, therefore it is quite important to reduce microprocessors power.

Many techniques for designing low power microprocessor have been proposed. In those there is a datapath width optimization technique[3] for low power. Analyzing statically variables at the RTL design stage, the minimum datapath width by which assure correct execution can be obtained by the technique. After that, redundant circuits are eliminated statically. This technique focuses on the minimum bits of variables. On the other hand our technique focuses on the minimum bits of each data which is a value of a variable. Run time analysis and optimization are features of our proposed technique. We use *active bits* and *inactive bits* which are defined as necessary and unnecessary bits of each data for computation, respectively. Some techniques using active bits have been studied [4,5]. Okuma et al. proposed a technique for low power data memory design by using active bits. Brooks et al. presented a low power arithmetic design technique by using active bits. We introduce a technique for low power instruction bus which is an important component in a processor based system. Their and our basic idea is to reduce redundant power consumption in inactive bits.

In CMOS circuit dynamic power for charging and discharging of internal node capacitances is dominant factor of total power consumption. The capacitances at I/O pins are orders of magnitude higher than internal capacitances. Therefore techniques for minimizing switching at external buses are quite important. The average power P is given by $P = 1/2 \cdot C \cdot V_{VDD}^2 \cdot f \cdot \alpha$, where C is the load capacitance, V_{DD} the supply voltage, f the clock frequency, and α the switching activity. If a technique promise to reduce P, without performance penalty, it will also reduce the energy consumption. Therefore it is preferable that any power minimization technique should take on no performance penalty. We focus on reducing α, because lowering α is effective way to reduce P especially off-chip buses.

Many encoding schemes to reduce switching activity of buses have been studied. For low power address buses many techniques[8,9,10] exploit characteristics that successive data have high correlation. In contrast on data buses each data has irregular value. In this case the bus-invert technique[7] works effectively. Since instruction buses have fixed forms, we reduce instruction buses power consumption by considering the semantics. In VLIW machines a technique[6] by which rearranges instruction sequences in off-chip memory is proposed for reducing switching activity during instruction fetches. This technique is static analysis and optimization. Our proposed technique reduces switching activity by dynamic active bits analysis and optimization using active bits. We present also active bits reduction techniques for more reducing power consumption. Operations and registers that appear more frequently are assigned shorter codes. This is a variable length coding. This assignment achieves reduction of total active bits in programs. For example, frequent values of register number are 4 and 5 in register fields. When the register numbers are transferred on the instruction bus, we use only 3 bits (active bits) from the least significant bit and remaining bits (inactive bits) are unused. If the register number 4 and 5 are rearranged register number 0 and 1, respectively, we only use 1 bit from the most significant bit on the bus.

This paper is organized as follows. Section 2 describes a definition of active bits and instruction bus. Section 3 introduces the encoding mechanisms by using active bits. Active bits reduction techniques for low power are discussed in section 4. Experimental results are presented in section 5. Section 6 concludes the paper.

2. Preliminaries

In this section we define active bits and describe characteristics of an instruction bus.

2.1. Active bits

We consider D that is an unsigned decimal number ($0 \le D \le 2^N - 1$), where N is a natural number. The number D can be expressed as follows:

$$D = \sum_{i=1}^{j} a_i \cdot 2^{i-1}$$

where $a_i \in \{0,1\}$ and $a_j = 1 (1 \le j \le N)$ for $D \ne 0$.

When we transfer the data on an N-bit width bus, the data is encoded with N bits as follows:

$$S = (0_N, 0_{N-1}, \ldots, 0_{j+1}, a_j, a_{j-1}, \ldots, a_2, a_1).$$

The number of bits of the encoded data is N all the time, however only j bits in total are essentially required to represent the D. All bits from least significant bit to the j bit are defined as *active bits*. On the other hand, *inactive bits* are defined as remaining bits after eliminating active bits (that is inactive bits are all bits from the ($j+1$)th-bit to the Nth-bit). The j which indicates the most significant bit in active bits is called as *active bitwidth* in this paper. We define a variable to indicate active bitwidth as A_{width}. Inactive bits are actually unnecessary, in consequence the bits cause redundant switching on buses. It is important to restrain redundant power consumption in inactive bits.

We introduce a procedure to obtain active bits from the S. In this paper, we assume that a representation of all data is an unsigned form. For the unsigned integers, inactive bits are continuous string of 0's from the most significant bit and the remaining bits are active bits.

2.2. Instruction bus

We assume that an instruction once instruction bus access is transferred. In this paper we use the SimpleScalar architecture. The SimpleScalar architecture[2] is derived from the MIPS-IV ISA[1]. Figure 1 shows the three instruction encoding of SimpleScalar instructions: register, immediate and jump formats. All instructions are 64 bits in length. However the 16 bits annote field and upper 8 bits of the opcode field are not used in this paper. In consequence we use only 40 bits from the least significant bit. Since the number of operation code is 109 in total, we can encode all instructions by using only 7 bits. But to apply 8 bits to opcode facilitates fast instruction decoding. The register fields are all 8 bits, to support extension of the registers to 256 integer and floating point registers. However only 5 bits are used for register identification. When we consider the semantics of instructions, the active bits in instructions are determined by each field.

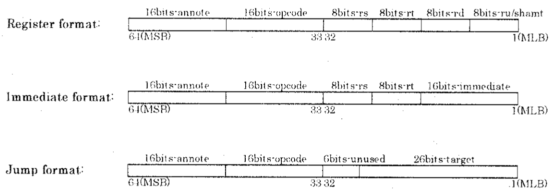

Figure 1: SimpleScalar architecture instruction formats

3. Encoding by using active bits

In this section, we present our encoding mechanism to reduce switching activity on instruction bus by using active bits. On instruction bus, active bits can be defined by each instruction format. For accurate definition of active bits, it is necessary to distinguish instruction formats, however for that purpose we need to prepare a format identification circuit. Since to make this circuit increases overhead such as power consumption and delay, we assume that the instruction has always five fields (the number of bits of all fields is 8) without considering immediate and target fields in immediate and jump formats. Hereafter we call the fields from the lowest field as f1, f2, f3, f4 and f5 in order (f5 constantly stands for opcode).

To suppress switching in inactive bits is our basic idea. Now we encode the source data S by using current data C on instruction bus as follows:

$$C = (c_N, c_{N-1}, .., c_i, .., c_2, c_1), c_i \in \{0,1\}.$$

Encoded data is given by

$$T = (t_N, t_{N-1}, .., t_i, .., t_2, t_1), t_i \in \{0,1\}.$$

The t_i is given by

$$t_i = \begin{cases} c_i, (i \geq A_{width} + 1) \\ a_i, (i \leq A_{width}) \end{cases}$$

In this way, we can suppress switching in active bits.

However we need extra bus to hold values of active bitwidth in addition to original bus. We define extra code E as follows:

$$E = (e_{E_{width}}, e_{E_{width}-1}, .., e_i, .., e_2, e_1)$$

where e_i is a value of ith bit from the least significant bit, and E_{width} indicates extra bus bitwidth .There are N+1 kinds of active bitwidth (one bit is utilized for active bitwidth 0). We adopt the binary encoding to encode A_{width}. When N is power of 2, E_{width} is $1 + \log_2 N$. On the other hand, when the number of elements to be encoded is N, E_{width} is $\log_2 N$. Since increasing extra bits cause additional power consumption and cost, we use active bits N-1 like active bits N (E_{width} is $\log_2 N$).

3.1. Active Bits Detection Granularity Control

It is important to improve the encoding for reducing extra bits because of increasing I/O pins and switching in extra bits. Using rough granularity for active bits detection, we can reduce extra bits. Initially instruction bus is divided into N_{seg} parts (N_{seg} is power of 2). S and Z are defined as source data and divided segments, respectively. The number of bits in each segment is equal ($N / N_{seg} = const. = \beta$) in this paper. We partition source data as follows:

$$S = ((0_N, ..., 0_{\beta(N-1)+1}), .., (0_{\beta(i-1)+1}, .., a_{A_{width}}, .., a_{\beta i}), .., (a_\beta, ..., a_1)).$$

When we define Z_i as

$$s_k \in \{0,1\}, Z_i = (s_{\beta(i-1)+1}, .., s_k, .., s_{\beta i}),$$

S can be written by

$$S = (Z_{N_{seg}}, Z_{N_{seg}-1}, ..., Z_2, Z_1).$$

When Z_i includes one and more active bits (a_k), we call the Z_i as *active segment*. If a segment is active segment, all bits in the segments are transferred directly. In contrary if a segment is not active segment, all bits in the segments hold current values.

Instead of active bitwidth we encode the number of active segment by using extra bits. There are N_{seg}+1 kinds of active segments. Smaller N_{seg} can reduce extra bits, because E_{width} is $1 + \log_2 N_{seg}$. For more extra bits reduction, we use active segment Z_{Nseg-1} like active bitwidth Z_{Nseg}. In consequence of the contrivance, E_{width} is $\log_2 N_{seg}$.

3.2. Embedding Approach

This subsection introduces a technique that information to indicate a boundary between active bits and inactive bits is embedded in transferred data, which is called as *embedding* approach. Using the last transferred data for a receiver, the embedding approach can be achieved. The procedure is as follows:

$$t_i = \begin{cases} c_i, (i > A_{width}) \\ \overline{c_i}, (i = A_{width}) \\ a_{i,} (i < A_{width}) \end{cases}$$

where the $\overline{c_i}$ bit is an inverted value of *k*th bit in the last transferred data. This technique is a encoding without extra bits. To obtain the boundary between active bits and inactive bits is possible only if continuity of all accesses is guaranteed. Therefore when this approach is used in the cases of multi receivers and transmitters bus system, the guarantee must be kept to. Note that more additional circuits for holding the last transferred data in a receiver are needed.

4. Active bits reduction technique

In the section 3, we proposed the switching count reduction techniques by using active bits dynamically. In this section we present a technique to achieve increased effectiveness of switching reduction. The principle of the technique in the section 3 is to reduce redundant switching in inactive bits. It is useful to increase inactive bits for high effective switching reduction. In consequence we aim to reduce active bits in data on instruction bus.

Figure 2 and 3 show frequency of appearance in each operation and register. An mpeg2play program with a 245k byte picture is adopted as sample application. High frequent 10 operations occupy 83.4% of total accesses. Moreover accesses of five frequent registers in total are 70.3% of all accesses. We aggressively utilize this frequent operations and registers. Using opcode and register code rearrangement, we reduce total active bits in programs. Operations and registers that appear more frequently are assigned shorter codes. This is a variable length coding. This assignment achieves reduction of total active bits in programs.

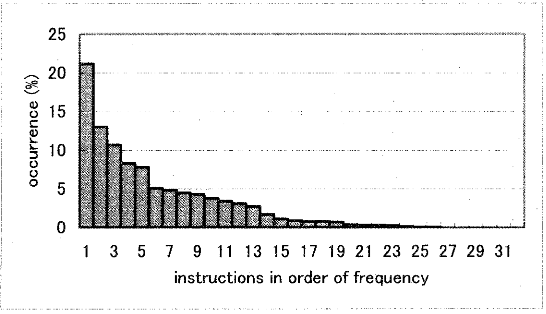

Figure 2: occurrence of operations (mpeg2play, 245k)

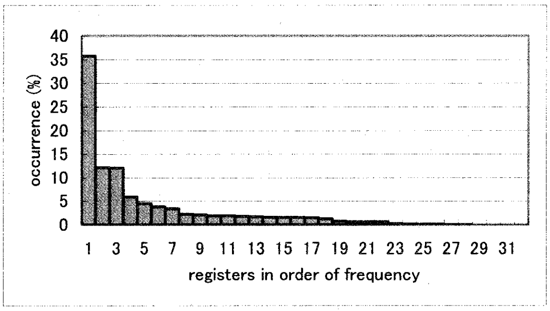

Figure 3: occurrence of registers (mpeg2play, 245k)

5. Experimental results

In this section, we present our experiments to reduce power consumption by using our proposed techniques. For our simulation, the SimpleScalar processor simulator[2] whose model has 40 bits width an instruction bus is employed. In the instruction formats there are useless bits, but the bits are prepared to extend instruction sets easily. For instance, under default condition the number of both integer and floating point registers is 32. 5 bits are enough to encode register assignment; however 8 bits are assigned in register fields. Since 3 bits from most significant are all 0, increasing switching is not caused in the bits of instruction bus.

Figure 4 shows comparison of switching count after applying proposed techniques on the instruction bus. We selected the mpeg2play program as an experimental application program, whose data size is 245k byte and instruction count is 373475424. We normalized the results of switching count. The most left side bar shows switching count in original case. Next four bars are results of applying the switching count reduction technique by using active bits without active bits reduction techniques. Another five bars and next five bars demonstrate the

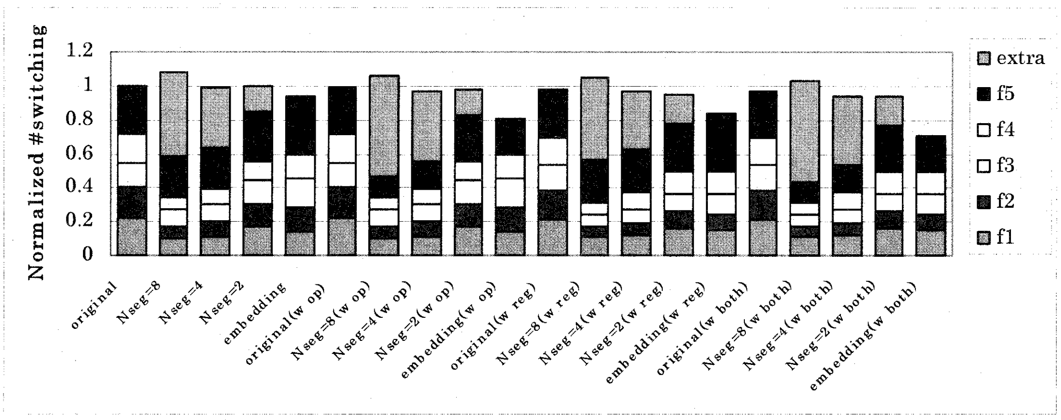

Figure 4: switching count in original and optimized instructions on instruction bus (mpeg2play, 245k)

results using opcode and register code rearrangement, respectively. The last five bars are the results using both active bits reduction techniques. When the basic technique is applied to the original bus by using active bits without active bits reduction techniques, a few switching count is reduced. In the case using active bits reduction techniques, we can achieve more switching count reduction. Up to 29.1% of switching count of original bus is decreased. In most fine granularity case, switching count increases more than the original bus. Figure 5 shows active bits in the original case and the case using active bits reduction techniques. The left bar indicates total bits when all instructions of the program are transferred on the bus. The middle bar and right bar are active bits in the original case and the case after reducing active bits. Using active bits reduction techniques, we can reduce about 25% of active bits of the original case.

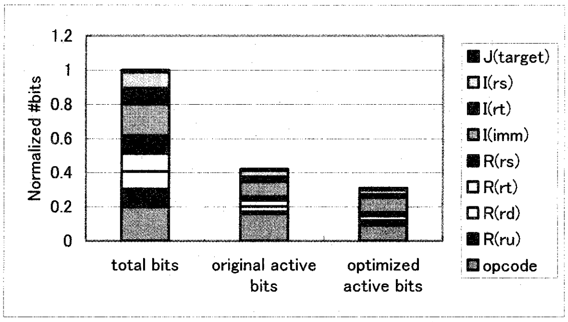

Figure 5: active bits in original instructions and optimized instructions (mpeg2play, 245k)

Several SPEC2000 benchmark programs[11] are used to verify effectiveness of our techniques. From SPECint2000 and SPECfp2000 benchmarks, gcc and mesa programs are adopted, respectively. Code sizes of the programs are both large compared to the other SPEC2000 benchmark programs (#instructions of gcc and mesa are 114970064 and 9146104, respectively). In figure 6 and 7 results are demonstrated. In case A, opcode and register assignments are customized to each program. In case B, the assignments are specified for mpeg2play. These results prove effectiveness of the techniques. Even if the instruction bus is customized for certain program by using our proposed techniques, the same effectiveness can be accomplished.

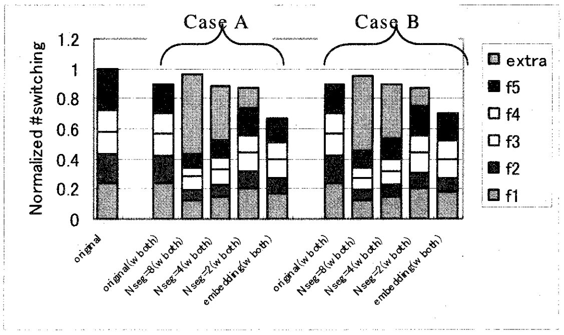

Figure 6: switching count on an instruction bus (gcc)

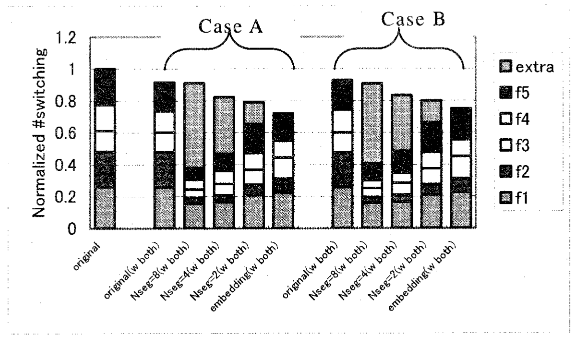

Figure 7: switching count on an instruction bus (mesa)

6. Conclusions

In this paper, we introduced a novel technique for low power instruction buses by using active bits with consideration of instruction semantics. Active bits reduction techniques without loosing information are also presented, which are based on variable length coding. Proposed techniques focus on essence of information and suppressing redundant switching of unessential parts. Experimental results demonstrate that our techniques are effective and up to 29.1% power saving for MPEG2. When different technique each fields is applied by considering characteristics of each field, more power reduction can be achieved. VLIW architectures require a high-bandwidth instruction fetch mechanism to supply multiple operations per cycle. Therefore using our proposed techniques, power consumption of VLIW machines in systems are quite reduced.

Since our techniques are available for not only instruction bus but also instruction memory, our future work is to design whole system by using active bits for low power with consideration of data semantics.

References

[1] Charles Price. *MIPS IV Instruction Set, revision 3.1*. MIPS Technologies, Inc., Mountain View, CA, January 1995.

[2] SimpleScalar version 2.0 user's guide. http://www.simplescalar.com/docs/users_guide_v2.pdf.

[3] H. Yamashita, H. Tomiyama, A. Inoue, F. N. Eko, T. Okuma, and H. Yasuura. Variable size analysis for datapath width optimization. In *Proceedings of Fifth Asian Pacific Conference on Hardware Description Language*, pages 69-74, July 1998.

[4] D. Brooks and M. Martonosi. Value-based clock gating and operation packing: dynamic strategies for improving processor power and performance. *ACM Transactions on Computer Systems*, 18(2):89-126, 2000.

[5] T. Okuma, Y. Cao, M. Muroyama, and H. Yasuura. Reducing access energy of on-chip data memory considering active data bitwidth. In *Proceedings of the 2002 international symposium on Low power electronics and design*, pages 88-91. ACM Press, 2002.

[6] C. Lee, J. K. Lee, and T. T. Hwang. Compiler optimization on instruction scheduling for low power. In *Proceedings of the 13th International Symposium on System Synthesis*, pages 55-60. IEEE Computer Society Press, 2000.

[7] M. R. Stan and W. P. Burleson. Bus-invert coding for low-power I/O. *IEEE Transaction on VLSI Systems*, 3(1):49-58, March 1995.

[8] H. Mehta, R. M. Owens, and M. J. Irwin. Some issues in glay code addressing. In *Proceedings of the 6th Great Lakes Symposium on VLSI*, pages 178-180, March 1996.

[9] E. Musoll, T. Lang, and J. Cortadella. Working-zone encoding for reducing the energy in microprocessor address buses. *IEEE Transaction on VLSI Systems*, 6(4):568-572, December 1998.

[10] P. R. Panda and N. D. Dutt. Reducing address bus transitions for low power memory mapping. In Proceedings of 1996 European Design and Test Conference, pages 63-67, March 1996.

[11] MIRV Benchmarks. http://www.eecs.umich.edu/mirv/benchmarks/benchmarks.html.