A Low Power VLIW Processor Generation Method by Means of Extracting Non-redundant Activation Conditions

Hirofumi Iwato, Keishi Sakanushi, Yoshinori Takeuchi, and Masaharu Imai
Graduate School of Information Science and Technology, Osaka University, 1-5 Yamadaoka, Suita, Osaka 565-0871, Japan
{h-iwato, sakanusi, takeuchi, imai}@ist.osaka-u.ac.jp

ABSTRACT

This paper proposes a low power VLIW processor generation method by automatically extracting non-redundant activation conditions of pipeline registers for clock gating. It is important for the best power reduction by clock gating to create control signals that can completely shut off redundant clock supplies for registers. In order to generate the control signals automatically, the proposed method utilizes high-level architecture information called Micro-Operation Descriptions, which describes a VLIW processor architecture. Exploiting the Micro-Operation Descriptions in a VLIW processor generation process, the proposed method automatically extracts the non-redundant activation conditions that can control clock gating to supply the minimum clocks to the pipeline registers. Using the non-redundant activation condition extraction, the proposed method achieves short calculation time and low area overhead; the proposed method can be applied to VLIW processor generation. Experimental results show that the VLIW processor generated with proposed method achieves power reduction about 60% compared to the non-clock-gated VLIW processor, and about 35% compared to the VLIW processor that is applied clock gating by PowerCompiler with negligible area overhead.

Categories and Subject Descriptors

B.6 [**Register-Transfer-Level Implementation**]: Design Aids—*Automatic synthesis*

General Terms

Algorithms, Design

Keywords

ASIP, Clock Gating, Low Power, VLIW Processor

1. INTRODUCTION

Modern embedded systems for portable applications need more computing power and less energy consumption. A VLIW processor is an answer to the demand since it can provide high computing power by exploiting instruction-level parallelism. When utilizing the VLIW processor as an embedded processor, it is necessary to simultaneously satisfy the tight constraints of area and energy consumption. For that purpose, Design Space Exploration (DSE) should be performed to determine the optimal architecture parameters of the VLIW processor [8] [11]. However, DSE is a time consuming task; an automatic VLIW processor generation method is proposed for the rapid DSE [9], because it can provide significant short design time and high design flexibility. Nevertheless, the traditional VLIW generation method places priority on minimizing area. To meet the recent demands for low power, a low-power specific VLIW processor generation method is desired.

There are many low power techniques through the whole design level. In gate-level design, two major techniques are familiar. Operand isolation [13] [4], which is one of the low power techniques, stabilizes input signals to a combinational resource when its result is not used by following circuits. Clock gating [10] is the another low power technique by means of inserting gates to clock lines of registers for cutting off redundant clock supply that causes unnecessary switchings inside the registers. Since an enormous amount of energy is consumed by registers in synchronous circuits [15], the clock gating appears to be a promising technique for energy reduction. As discussed previously, clock gating can selectively shut off excess clocks, i.e., clock gating can selectively supply only necessary clocks. Accordingly, it is important for the best energy reduction by clock gating to extract non-redundant activation conditions of registers in a circuit. The non-redundant activation conditions indicate that registers need clocks only when the conditions are satisfied.

With respect to the VLIW processor, there is a large number of pipeline registers (over several hundred) in its large-scale data path. Furthermore, the number of the pipeline registers rapidly increases as the number of parallel issue increases, hence a large amount of energy is dissipated by the pipeline registers [5]. For this reason, this paper focuses on energy reduction of pipeline registers in VLIW processors. It is an unrealistic way to manually extract the non-redundant activation conditions of the pipeline registers, because it takes long-term design period and also it is error-prone. For extracting the non-redundant activation conditions automatically, some approaches have been introduced [1] [2] [3] [14]. However, they cannot be applied to large-scale designs as the VLIW processors because of their limitations due to analyzing complex RTL designs.

CODES+ISSS'07, September 30–October 3, 2007, Salzburg, Austria.

In order to automatically derive the non-redundant activation conditions of the VLIW processors, this paper proposes a new approach based on high level architecture information namely a Micro-operation description (MOD) [7]. MODs specify a VLIW architecture including architecture parameters such as a number of parallel issues, instruction formats, and behavior of each instruction. By employing the MOD for generating VLIW processor, the proposed method can derive the non-redundant activation conditions of the pipeline registers in the processor generation process, analyzing neither RTL descriptions nor netlists.

The rest of this paper is organized as follows. Section 2 introduces some state-of-the-art related researches. Section 3 explains MOD and Section 4 explains VLIW processor model. Section 5 proposes the low power VLIW processor generation method. Section 6 shows experimental results, and finally, Section 7 summarizes this paper.

2. RELATED WORK

Several automatic clock gating insertion methods/tools are currently available. PowerCompiler is the most widely known commercial tool, which automatically inserts gates to clock lines of registers. However, PowerCompiler does not extract the non-redundant activation conditions of the registers, that is to say, efficiency of clock gating by PowerCompiler rests on the shoulders of designers. PowerCompiler forces designers to manually derive the non-redundant activation conditions from complex RTL designs for further power reduction. Nevertheless, manual condition extraction is a heavy time consuming task, because VLIW processors contains several hundred pipeline registers; it is not suitable for DSE. Automated extraction of the non-redundant activation conditions is strongly required for clock gating.

A finite state machine based clock gating method [2] extracts non-redundant activation conditions of registers by analyzing finite state machines in a circuit. Obviously, to use this method, the circuit must contain the finite state machines and be controlled by them. Since the finite state machine is not suitable for pipeline processors, it cannot be applied to VLIW processor generation.

Observability Don't Care (ODC) based clock gating method discussed in [1] can derive the non-redundant activation conditions of registers by ODC calculation. At present, this approach appears to be the most powerful clock gating method because of its high scalability and applicability. Unfortunately, ODC calculation takes too long time to entirely apply the large-scale circuitry such as the VLIW processors. To complete the calculation in realistic time, limit of calculation time is introduced in [1]. In addition, the ODC-based clock gating extraction causes an enormous area overhead due to duplicating circuits for creating gating signals. To calculate the non-redundant activation conditions for VLIW processor, a more suitable condition extraction method is still required.

The operand isolation approach exploiting high-level architecture information is addressed in [4]. In this approach, high-level architecture information called Architecture Description Language (ADL) is employed to find idle conditions of resources in a circuit. Although the use of the operand isolation techniques is discussed in this work, the use of clock gating with high-level architecture information has not been discussed yet.

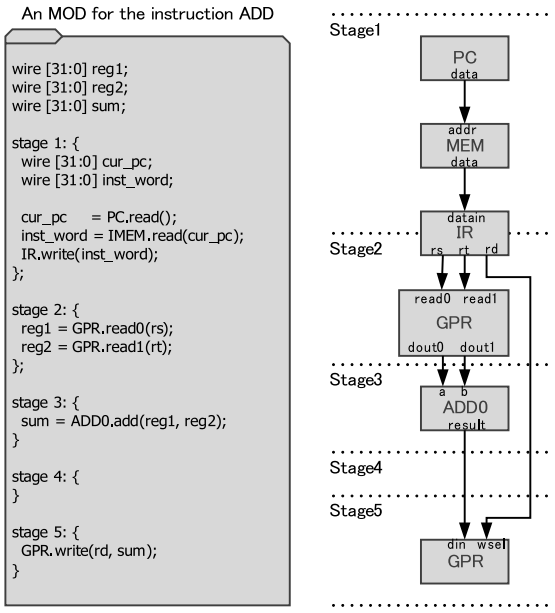

Figure 1: The MOD of an operation ADD

3. MICRO-OPERATION DESCRIPTION

An MOD [7] denotes an architecture of an operation. The operation is an executable unit such as arithmetic operations. Figure 1 is an example of an operation ADD, which performs arithmetic addition. The left side description is the MOD of ADD and the right side diagram is the corresponding architecture. The MOD describes signal connections between resource ports. In Figure 1, the two data reg1 and reg2 from the register file GPR at stage 2 are input to the adder ADD0 at stage 3, then the outcome is stored in the GPR at stage 5. The data rs, rt, rd stand for register indexes from the instruction register IR. The MEM is a memory access unit and the PC is a program counter. In this way, designers can briefly specify the architecture of each operation.

The proposed method automatically generates a VLIW processor from MODs and detects non-redundant activation conditions for the pipeline registers automatically.

4. VLIW PROCESSOR MODEL

The VLIW processor dispatching is modeled as following three important concepts: slot, operation group, and resource group. A VLIW instruction consists of multiple operations that can be issued in parallel from each slot. The slot is the number of parallel issue of the VLIW processor. The operation group is a set of operations that have the same characteristic on dispatching. The resource group is a set of hardware resources that can execute several operation groups. The issued operations are executed in the corresponding resource groups. The relations (illustrated as directed arrows in Figure 2) between the resource groups and the slots indicate that the operations issued from each slots can be performed on the related resource groups. The relations between resource groups and operation groups indicate that the operations categorized in the operation group can be performed on the related resource groups. The MOD of

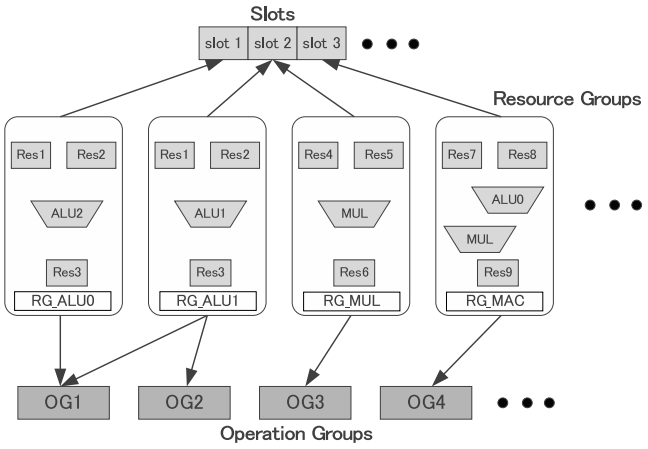

Figure 2: A VLIW processor model

each operation is described as a pair of an operation and an operation group.

Figure 2 illustrates a dispatching model of a VLIW processor [9]. In Figure 2, the operations categorized in the operation group $OG1$ can be performed on the resource groups RG_{ALU0} and RG_{ALU1}, and can be issued from $slot1$ and $slot2$ simultaneously.

5. VLIW PROCESSOR GENERATION

The VLIW processor generation consists of two parts. The first part is data path construction consisting of four procedures: Resource Connection Graph (RCG) extraction, RCG merging, signal conflict resolution, and pipelining. First, an RCG is extracted from an MOD of each operation. The RCG represents a data path of the corresponding operation. Then all RCGs are merged by the second procedure, RCG merging, to construct a prototype of the VLIW processor data path. Finally, multiplexers and pipeline registers are inserted in appropriate locations by the following two procedures. At each procedure, the generation method retrieves some conditions which is used to generate steering signals and resource control signals in the next part. The second part is the controller construction. Every control signals of resources in the VLIW processor are generated using the conditions obtained in the data path construction part.

For maximizing power reduction by clock gating, non-redundant activation conditions of pipeline registers are necessary. To this end, the proposed method calculates the non-redundant activation conditions in the data path construction part. Then, in the next controller construction part, the proposed method generates gating signals, which are control signals for clock gating, using the non-redundant activation conditions.

As mentioned in [10] and [12], selecting clock gating circuit scheme is important for reduction of design difficulty and implementation overhead. Despite its area overhead, the proposed method uses a flip-flop-based gating scheme because of its capability of blocking glitch noises causing incorrect register activation. The flip-flop-based gating circuits are inserted in all registers (not only pipeline registers) in the generated VLIW processor.

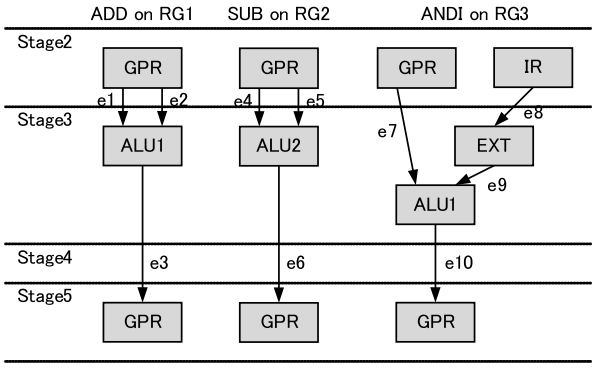

Figure 3: Extracted RCGs

5.1 CONDITION EXTRACTION

In this section, the non-redundant activation conditions of the pipeline registers are calculated in the following data path constructing procedures.

5.1.1 RCG extraction

RCGs are extracted by means of analyzing MODs of operations. An RCG is represented by a directed graph $G_{ope}(R_{ope}, E_{ope})$ where $ope \in O$, O represents a set of all operations, $R_{ope} \in RG$ is a set of resources used by an operations ope, $RG = \{R_i | i = 1, 2, ..., N_{RG}\}$ is a set of all resource groups, $N_{RG} = |RG|$, $R_i \subseteq R$ is a set of resource groups, R is a set of all resources, $E_{ope} = \{(o, i) | o, i \in P\}$ is a set of directed edge, representing data transfers between output ports o to input port i, and P is a set of all resource ports described as the union of all resource ports.

A set of conditions $Cond_e$ for each data transfer $e \in E_{ope}$ is retrieved in RCG extraction. $Cond_e$ is described as

$$Cond_{e \in E_{ope}} = \{(ope, R_{ope}) \,|\, ope \in I, R_{ope} \in RG\}. \quad (1)$$

The $Cond_e$ indicates that a data transfer e is executed when an operation ope on a resource group R_{ope} is decoded.

Figure 3 shows a simple example of extracted RCGs. For the sake of clarity, ports of the RCGs are omitted. In the example, there are three RCGs corresponding to ADD on RG_1, SUB on RG_2, and $ANDI$ on RG_3 respectively. GPR, IR, EXT, $ALU1$, and $ALU2$ are resources and edges labeled as $e1$ to $e9$ are data transfers. Here, sets of conditions for the data transfers in Figure 3 are retrieved in the form of eq.(1) as follows:

$$\begin{aligned}
Cond_{e1} &= \{(ADD, RG_1)\} \\
Cond_{e2} &= \{(ADD, RG_1)\} \\
Cond_{e3} &= \{(ADD, RG_1)\} \\
Cond_{e4} &= \{(SUB, RG_2)\} \\
Cond_{e5} &= \{(SUB, RG_2)\} \\
Cond_{e6} &= \{(SUB, RG_2)\} \\
Cond_{e7} &= \{(ANDI, RG_3)\} \\
Cond_{e8} &= \{(ANDI, RG_3)\} \\
Cond_{e9} &= \{(ANDI, RG_3)\} \\
Cond_{e10} &= \{(ANDI, RG_3)\}.
\end{aligned}$$

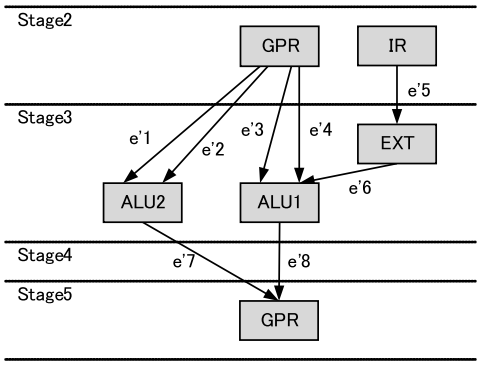

Figure 4: A Unified RCG

5.1.2 RCG merging

After all RCGs are extracted, the RCGs are merged into a unified RCG $G' = (R', E')$. R' and E' are calculated as follows:

$$R' = \bigcup_{ope \in O} R_{ope}$$
$$E' = \bigcup_{ope \in O} E_{ope}.$$

Since the data transfers E are merged into E', the conditions $Cond_e$ of all extracted RCGs are also merged. The new conditions of the data transfers $Cond'_e$ are calculated as

$$Cond'_{e' \in E'} = \bigcup_{ope \in O, e \in E_{ope}, e' = e} Cond_{e \in E_{ope}}. \quad (2)$$

The three operations in the previous example in Figure 3 are merged into the unified RCG illustrated as Figure 4. Conditions of the data transfers $e'1$ to $e'8$ are newly calculated as follows:

$$\begin{aligned}
Cond_{e'1} &= \{(SUB, RG_2)\} \\
Cond_{e'2} &= \{(SUB, RG_2)\} \\
Cond_{e'3} &= \{(ADD, RG_1), (ANDI, RG_3)\} \\
Cond_{e'4} &= \{(ADD, RG_1)\} \\
Cond_{e'5} &= \{(ANDI, RG_3)\} \\
Cond_{e'6} &= \{(ANDI, RG_3)\} \\
Cond_{e'7} &= \{(SUB, RG_2)\} \\
Cond_{e'8} &= \{(ADD, RG_1), (ANDI, RG_3)\}.
\end{aligned}$$

The unified RCG is a prototype of the data path of the VLIW processor. Then multiplexers and pipeline registers are inserted in the following procedures.

5.1.3 Signal conflict resolution

Multiplexers are inserted to resolve signal conflicts occurring in the unified RCG G' such as $e'7$ and $e'8$ in Figure 4. We describe the RCG after the multiplexers are inserted as $G'' = (R'', E'')$.

A multiplexer is inserted in front of an input port that has multiple destination of data transfers. Here, a set of data transfers conflicting at an input port i is described as

$$ECI(i) = \{(o', i') \mid o' \in P_{out},\ i' \in P_{in},\ (o', i') \in E',\ i' = i\}.$$

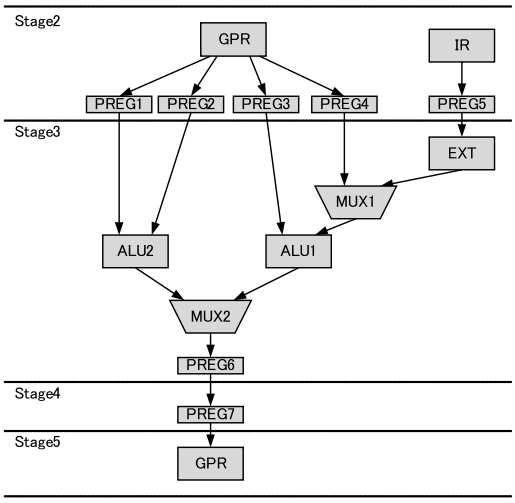

Figure 5: A Constructed Data Path

Since the multiplexers are inserted, signal connections change. Therefore, recalculating the conditions of the data transfers E'' is needed. A set of conditions of data transfer $e'' \in E''$, is described as follows:

$$Cond''_{e''} = \begin{cases} \bigcup_{e' = ECI(dest(e''))} Cond'_{e' \in E'} & \text{if } e'' \in E_{MR} \\ Cond'_{e' \in E'} \text{ such that } dest(e') = dest(e'') & \text{if } e'' \in E_{RM} \\ Cond'_{e' \in E'} \text{ such that } e' = e'' & \text{otherwise} \end{cases} \quad (3)$$

where $dest(e)$ is a destination port of a data transfer e, E_{MR} is a set of data transfers that connect a multiplexer to a resource, and E_{RM} is a set of data transfers that are selected by a multiplexer.

5.1.4 Pipelining

For pipelining G'', pipeline registers are required for data transfers crossing pipeline stage boundaries. A location of a pipeline register $p = (o, n)$ can be described as a pair of output port o and stage number n where the pipeline register is placed, since one pipeline register is shared by some data transfers from o. A set of data transfers connected to the same output port o is

$$ECO(o) = \{(o'', i'') \mid o'', i'' \in P,\ (o'', i'') \in E'',\ o'' = o\}.$$

In the $ECO(o)$, a set of data transfers crossing stage boundaries is

$$ECO_{CROSS}(o) = \{(o', i') \mid stage(o') < stage(i'), \text{ for all } (o', i') \in ECO(o)\}$$

where $stage(x)$ represents a pipeline stage number to which a port x belongs.

In the G'', a set of data transfers that cross stage boundaries is

$$E''_{CROSS} = \bigcup_{o \in P_{out}} ECO_{CROSS}(o).$$

Finally, a set of pipeline registers $PREG$ is obtained as

$$PREG = \bigcup_{(o,i)\in E''_{CROSS}} \{p \mid p=(o,n), stage(o) \le n < stage(i), n \in N\}. \quad (4)$$

Figure 5 depicts the data path after inserting multiplexers and pipeline registers in the unified RCG in Figure 4. The pipeline registers $PREG1$ to $PREG7$ are inserted in appropriate points.

Here, the non-redundant activation conditions of the inserted pipeline registers are calculated. The non-redundant activation conditions for the pipeline registers are derived from the conditions of the data transfers calculated by eq.(3). Since the pipeline registers are shared by some data transfers, a set of non-redundant activation conditions $Active_p$ of a pipeline register $p=(o,n)$ is calculated as

$$Active_p = \bigcup_{(o,n)=p,\ (o,i)\in ECO_{CROSS}(o), stage(i)>n} Cond''_{(o,i)}. \quad (5)$$

For the pipeline registers in Figure 5, the activation conditions $Active_p$ are calculated as follows:

$$\begin{aligned}
Active_{PREG1} &= \{(SUB, RG_2)\} \\
Active_{PREG2} &= \{(SUB, RG_2)\} \\
Active_{PREG3} &= \{(ADD, RG_1), ANDI, RG_3)\} \\
Active_{PREG4} &= \{(ADD, RG_1)\} \\
Active_{PREG5} &= \{(ANDI, RG_3)\} \\
Active_{PREG6} &= \{(ADD, RG_1), (SUB, RG_2), (ANDI, RG_3)\} \\
Active_{PREG7} &= \{(ADD, RG_1), (SUB, RG_2), (ANDI, RG_3)\}.
\end{aligned}$$

Thus the activation conditions for pipeline registers are calculated. In the next section, the gating signals for pipeline registers are discussed.

5.2 GATING SIGNAL GENERATION

By employing the non-redundant activation conditions given by eq.(5), a gating signal en_p for each pipeline register p is described as

$$en_p = \overline{stall_{stage(p)}} \wedge \left(\bigvee_{(ope,R_{ope})\in Active_p} decoded(ope, R_{ope}) \wedge active(R_{ope}) \right) \quad (6)$$

where $stage(p)$ represents stage number to which the pipeline register p belongs. $stall_n$ stands for a request for pipeline stall caused by several situations, e.g., structural hazard, multi-cycle operation. $decoded(ope, R_{ope})$ is a function that returns true when an operation in a instruction register is successfully decoded as operation ope on resource group R_{ope}. $active(R_{ope})$ is also a function that returns true when a VLIW instruction pattern is valid.

Using eq.(6), gating signals of the pipeline registers in the VLIW data path illustrated in Figure 5 can be calculated as follows:

$$\begin{aligned}
en_{PREG1} &= \overline{stall_2} \wedge (decoded(SUB,2) \wedge active(RG_2)) \\
en_{PREG2} &= \overline{stall_2} \wedge (decoded(SUB,2) \wedge active(RG_2)) \\
en_{PREG3} &= \overline{stall_2} \wedge (decoded(ADD,1) \wedge active(RG_1) \\
&\quad \vee decoded(ANDI,3) \wedge active(RG_3)) \\
en_{PREG4} &= \overline{stall_2} \wedge (decoded(ADD,1) \wedge active(RG_1)) \\
en_{PREG5} &= \overline{stall_2} \wedge (decoded(ANDI,3) \wedge active(RG_3)) \\
en_{PREG6} &= \overline{stall_3} \wedge (decoded(ADD,1) \wedge active(RG_1) \\
&\quad \vee decoded(SUB,2) \wedge active(RG_2) \\
&\quad \vee decoded(ANDI,3) \wedge active(RG_3)) \\
en_{PREG7} &= \overline{stall_4} \wedge (decoded(ADD,1) \wedge active(RG_1) \\
&\quad \vee decoded(SUB,2) \wedge active(RG_2) \\
&\quad \vee decoded(ANDI,3) \wedge active(RG_3)).
\end{aligned}$$

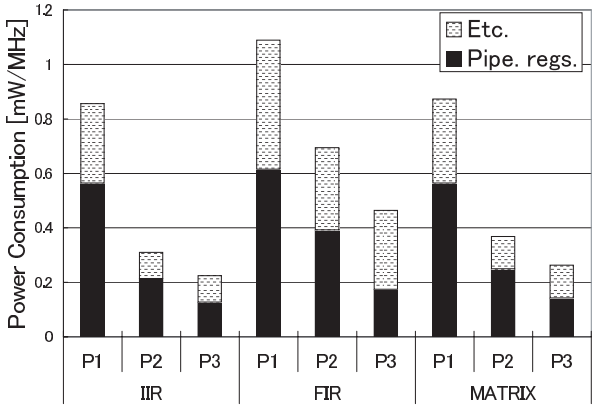

Figure 6: Energy comparison when executing various programs, fixing parallel issue number to 2

6. EXPERIMENTAL RESULT

In order to confirm the effectiveness of the proposed method, we carried out two experiments using DLX [6] instruction-set architecture. We designed following three version of VLIW architectures and measured power consumption, area, and delay of them.

Processor 1 A VLIW processor generated by traditional generation method (non-clock-gated VLIW processor)

Processor 2 A VLIW processor obtained by applying PowerCompier to the Processor 1 (traditional clock gating)

Processor 3 A VLIW processor generated by proposed generation method (proposed method)

We used DSPstone benchmark programs and compiled them with GCC based compilers that are adapted to each architecture by hand. The generated VLIW processors were synthesized with DesignCompiler employing a $0.14\mu m$ CMOS technology library.

The first experimental results are shown in Tables 1 and 2. Table 1 shows power comparison of the three version of VLIW processors, increasing parallel issue number (slot) from 2 to 4 for each. Note that the 2-slot processor contains

Table 1: Detailed energy comparison when changing parallel issue number (slot)

	Processor 1 (non-clock-gated)			Processor 2 (traditional clock gating)			Processor 3 (proposed method)		
	Pipe. regs. [mW/MHz]	Etc. [mW/MHz]	Total [mW/MHz]	Pipe. regs. [mW/MHz]	Etc. [mW/MHz]	Total [mW/MHz]	Pipe. regs. [mW/MHz]	Etc. [mW/MHz]	Total [mW/MHz]
slot 2	0.20536	0.26846	0.47381	0.14209	0.12912	0.27121	0.05507	0.12747	0.18254
slot 3	0.37894	0.33264	0.71158	0.26193	0.17402	0.43596	0.10608	0.17590	0.28198
slot 4	0.58947	0.39544	0.98491	0.40850	0.21930	0.62780	0.17021	0.22775	0.39796

Table 2: Area and critical path delay comparison

	Processor 1		Processor 2		Processor 3	
	Area [cells]	Delay [ns]	Area [cells]	Delay [ns]	Area [cells]	Delay [ns]
2 slots	70279	9.77	63588	8.18	63857	8.08
3 slots	113083	10.59	101889	9.08	102345	8.4
4 slots	168928	10.26	152517	10.82	153365	9.98

74 pipeline registers, 3-slot processor does 145, and 4-slot processor does 238; they are very large-scale designs. As observed in Table 1, the total energy consumption of Processor 3 is reduced approximately 60% compared to Processor 1 on every slot number. Compared to Processor 2, the Processor 3 reduces up to 36% of total power consumption. With respect to the power reductions of pipeline register, the Processor 3 achieves roughly 70% of power reduction compared to Processor 1. Furthermore, Processor 3 achieves roughly 60% of power reduction compared to Processor 2. Besides, power consumptions of other resources (Etc. in Table 1) in the data path are also reduced by applying the proposed method. This is because less data transitions of pipeline registers cause less signal switchings in the following resources such as ALU. Table 2 shows area and critical-path delay. As seen in Table 2, the area overhead of Processor 3 against Processor 2 is negligible. This is because the gating signals are derived from already decoded signals of non-clock-gated Processor 1. Furthermore, critical-path delay overheads are also negligible. As shown in Table 2, applying clock gating results in area reduction (from Processor 1 to the others). It is not extraordinary reduction. This is because multiplexers inside registers are removed when clock gating is applied. Area reduction due to applying clock gating is often observed.

The second experimental result is shown in Figure 6. We executed three programs, IIR, FIR, and MATRIX from DSPstone benchmark, fixing slot number to 2. In either case, energy consumption of Processor 3 (P3) is lower than the other processors (Processor 1 as P1, Processor 2 as P2). The differences of energy reduction ratio is due to the differences of data path utilization ratio of the individual programs.

7. CONCLUSION

The low power VLIW processor generation method was proposed in this paper. The proposed method automatically extracts the non-redundant activation conditions of the pipeline registers in generated VLIW processors and completely shut off excess clock supplies to the pipeline registers with clock gating. The experimental results showed significant power reduction of the pipeline registers in the generated VLIW processors, comparing with a traditional clock gating method. Area and delay overheads are confirmed to be negligible. Our future work is aimed at the simultaneous use of clock gating and operand isolation. In addition, the proposed method is not limited to VLIW architecture; extension of the proposed method to other architecture such as super scalar processors is also our future work.

8. REFERENCES

[1] P. Babighian, L. Benini, and E. Macii. A scalable algorithm for RTL insertion of gated clocks based on ODCs computation. *IEEE Trans. Computer-Aided Design*, 24(1):29–42, Jan. 2005.

[2] L. Benini and G. D. Micheli. Transformation and synthesis of FSMs for low-power gated-clock implementation. In *Proceedings of the ISLPED '95*, pages 21–26, 1995.

[3] L. Benini, G. D. Micheli, E. Macii, M. Poncino, and R. Scarsi. Symbolic synthesis of clock-gating logic for power optimization of synchronous controllers. *ACM Trans. Des. Autom. Electron. Syst.*, 4(4):351–375, 1999.

[4] A. Chattopadhyay, B. Geukes, D. Kammler, E. M. Witte, O. Schliebusch, H. Ishebabi, R. Leupers, G. Ascheid, and H. Meyr. Automatic ADL-based operand isolation for embedded processors. In *Proceedings of the DATE '06*, pages 600–605, 2006.

[5] D. Duarte, N. Vijaykrishnan, and M. Irwin. A clock power model to evaluate impact of architectural and technology optimizations. *IEEE Tran. VLSI*, 10(6):844–855, Dec. 2002.

[6] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, Inc., California, 1990.

[7] M. Itoh, Y. Takeuchi, M. Imai, and A. Shiomi. Synthesizable HDL Generation for Pipelined Processors from a Micro-Operation Description. *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences*, E83-A(3):394–400, Mar. 2000.

[8] M. F. Jacome, G. de Veciana, and V. Lapinskii. Exploring performance tradeoffs for clustered VLIW ASIPs. In *Proceedings of the ICCAD '00*, pages 504–510, 2000.

[9] Y. Kobayashi, S. Kobayashi, K. Okuda, K. Sakanushi, Y. Takeuchi, and M. Imai. Synthesizable HDL generation method for configurable VLIW processors. In *Proceedings of the ASPDAC '04*, pages 842–845, June 2004.

[10] T. Lang, E. Musoll, and J. Cortadella. Individual flip-flops with gated clocks for low power datapaths. *IEEE Trans. Circuits Syst. II*, 44(6):507–516, June 1997.

[11] B. Middha, A. Gangwar, A. Kumar, M. Balakrishnan, and P. Ienne. A Trimaran based framework for exploring the design space of VLIW ASIPs with coarse grain functional units. In *Proceedings of the ISSS '02*, pages 2–7, 2002.

[12] M. Mueller, A. Wortmann, S. Simon, M. Kugel, and T. Schoenauer. The impact of clock gating schemes on the power dissipation of synthesizable register files. In *Proceedings of the ISCAS '04*, 2004.

[13] M. Munch, B. Wurth, R. Mehra, J. Sproch, and N. Wehn. Automating RT-level operand isolation to minimize power consumption in datapaths. In *Proceedings of the DATE '00*, pages 624–633, 2000.

[14] M. Ohnishi, A. Yamada, H. Noda, and T. Kambe. A method of redundant clocking detection and power reduction at RT level design. In *Proceedings of the ISLPED '97*, pages 131–136, 1997.

[15] M. Pedram. *Power Aware Design Methodologies*. Kluwer Academic Publishers, Norwell, MA, USA, 2002.