IPSJ Transactions on System LSI Design Methodology Vol. 3 222–233 (Aug. 2010)

Regular Paper

A Low-power ASIP Generation Method by Extracting Minimum Execution Conditions

Hirofumi Iwato,[†1] Keishi Sakanushi,[†1] Yoshinori Takeuchi[†1] and Masaharu Imai[†1]

This paper proposes a low-power ASIP generation method by automatically extracting minimum execution conditions of pipeline registers for clock gating. For highly effective power reduction by clock gating, it is important to create minimum execution conditions, which can shut off redundant clock supplies for registers. To automatically extract the conditions, our proposed method employs micro-operation descriptions (MODs) that specify ASIP architecture. Utilizing MODs through the ASIP generation processes, our proposed method automatically extracts the minimum execution conditions. Experimental results show that the power consumption of the pipeline registers in ASIPs generated with the proposed method is reduced about 80% compared to ASIPs that are not clock gated, and about 60% compared to ASIPs that are clock gated by Power Compiler with negligible delay and area overhead.

1. Introduction

Modern portable embedded systems are required to provide high computing power with low energy consumption. Application Specific Instruction-set Processors (ASIPs) answer to such conflicting requirements. In recent years, Very Long Instruction Word (VLIW) type ASIPs (VLIW ASIPs)[1] have been proposed for high performance application domains because they can provide high computing power by special instruction set architecture which exploits instruction level parallelism. When designing VLIW ASIPs, simultaneously satisfying the tight constraints of delay, area, and power consumption is required. For this purpose, design space exploration should be performed to determine the optimal architecture parameters of VLIW ASIPs[2],[3]. One of the main concerns is that design space exploration is time consuming. For rapid design space exploration, an automatic VLIW ASIP generation method that achieves significant short design time has been proposed[1]. However, the traditional VLIW ASIP generation method places a priority on minimizing area and delay. For reducing power consumption, it is important to generate VLIW ASIPs with low power techniques.

There are many low power techniques through all design levels. In Register Transfer Level (RTL), clock gating is familiar[4]. Clock gating cuts off the clock supplies causing unnecessary switching inside the registers based on the gating conditions. Generally, there are many pipeline registers in the large-scale data path of a VLIW ASIP. Furthermore, the number of pipeline registers rapidly increases when widening the parallel issue or deepening the pipeline; thus a large amount of energy is dissipated in the pipeline registers[5]. For this reason, clock gating on pipeline registers is expected to be effective for VLIW ASIP generation.

Using only the interlocking conditions of pipeline registers is a trivial method to apply clock gating to pipeline registers in a VLIW ASIP. The control signal of a pipeline register p in a generated VLIW ASIP is described as:

$$en_p = \overline{stall_{stage_p}}, \tag{1}$$

where $stage_p$ represents the stage number to which p belongs. $stall_n$ stands for a condition of a pipeline interlock caused by several situations, e.g., structural hazard or multi-cycle operation. Since the aim of traditional VLIW ASIP generation[1] is to design high-speed and small-area ASIPs; the form of signal (1) is simplified as much as possible. By using control signal (1) for clock gating, clocks are supplied to all pipeline registers when the pipeline is not interlocked. However, applying control signal (1) has less effect on power reduction because the pipeline rarely stalls in VLIW applications, e.g., DSP. To reduce power consumption in such cases, it is necessary to consider the fact that a large portion of the data path is idle during instruction execution even if the pipeline is not interlocked. Clocks must be supplied to only necessary pipeline registers for power reduction.

In order to supply clocks to the only necessary pipeline registers, we introduce *minimum execution conditions* for control signal generation. The minimum execution conditions of a resource are conditions for clock supply and represented by

†1 Osaka University

a set of instructions which use the resource for their execution, i.e., the resource is used only when the instructions are executed. Using the minimum execution conditions for the control signals of clock gating, clock supplies are limited to only necessary pipeline registers; therefore, power consumption is reduced.

Manual extraction of the minimum execution conditions is not practical because it requires a long-term design period and is error-prone. An automated extraction method of the minimum execution conditions is strongly required. To automatically obtain the minimum execution conditions, two major approaches are known: forward [6),7)] and backward [8)]. The forward approach extracts the minimum execution conditions through high-level synthesis processes using high-level architecture information. On the other hand, the backward approach extracts the conditions by analyzing low-level designs. A merit of the backward approach is that it can be widely applied to arbitrary circuits. However, they need long calculation time and involve large area overhead due to the analysis of their complex design. Therefore, the forward approach is suitable for VLIW ASIP generation. Obviously, a forward approach must be designed for each high-level synthesis method. With respect to the VLIW ASIP generation method, an automated extraction method of minimum execution conditions is not known yet.

This paper proposes a VLIW ASIP generation method to generate low-power VLIW ASIPs by automatically extracting minimum execution conditions for clock gating. The proposed method extracts the conditions of pipeline registers based on the forward approach using architecture description language, namely, Micro-Operation Description (MOD) [9)], which specifies the high-level architecture of a VLIW ASIP. By employing the MODs, the proposed method can extract the minimum execution conditions of the pipeline registers through the VLIW ASIP generation processes, analyzing neither RTL descriptions nor netlists.

The rest of this paper is organized as follows. Section 2 introduces the state-of-the-art related researches, and Section 3 explains traditional VLIW ASIP generation. Section 4 proposes a method for extracting the minimum execution conditions. Section 5 shows experimental results, and finally, Section 6 summarizes this paper.

2. Related Work

Several automatic clock gating insertion methods and tools are currently available. Power Compiler [10)], which is the most widely known commercial tool, automatically inserts gates into the clock lines of registers. However, it does not extract the minimum execution conditions of the registers; that is, the efficiency of clock gating by Power Compiler rests on the shoulders of designers. Power Compiler forces designers to manually derive the minimum execution conditions from complex RTL designs for additional power reduction. Manual extraction of the conditions is very time consuming, because VLIW ASIP contains several hundred pipeline registers; it is not suitable for design space exploration. Automated extraction of the minimum execution conditions is strongly required.

A clock gating method based on finite state machines [6)] extracts the minimum execution conditions of registers by analyzing the finite state machines. To use this method, the circuit must obviously contain the finite state machines and be controlled by them. Since the finite state machines are not suitable for pipeline processors, it cannot be applied to VLIW ASIP generation.

A clock gating method based on the Observability Don't Care (ODC) [8)] can derive the minimum execution 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 to completely apply such large-scale circuitry as VLIW ASIPs. To complete the calculation in practical time, a calculation time limit is introduced in the method. In addition, ODC-based clock gating extraction causes an enormous area overhead due to duplicating the circuits to create gating signals. A more suitable extraction method is still required to calculate the minimum execution conditions for VLIW ASIPs.

The operand isolation approach that exploits high-level architecture information is also addressed [7)]. In this approach, high-level architecture information called Architecture Description Language (ADL) is employed to find the idle conditions of resources in a circuit. Although using operand isolation techniques is discussed in this work, using clock gating with high-level architecture information has not been discussed yet.

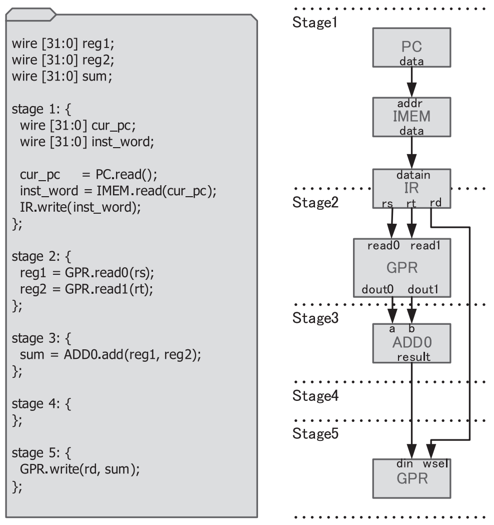
3. Basic of VLIW ASIP Generation

A VLIW ASIP is generated based on MODs and a VLIW ASIP model [1],[11]. In this section, MODs, the VLIW ASIP model, and the VLIW ASIP generation flow are described.

3.1 Micro-operation Description

The architecture of a VLIW ASIP is specified with MODs. An MOD is identified as a pair of an operation *ope* and a resource group *rg*: $m = (ope, rg)$. A set of all MODs specified by designers are represented by M. The MOD specifies the architecture of operation *ope* that is executed on resource group *rg*. An operation is a minimum executable unit such as an arithmetic operation, and a resource group is a set of hardware resources necessary for the operation. The architecture specified by an MOD can be converted to a Resource Connection Graph (RCG), which represents the data path of the corresponding operation in the form of the connections of resource ports.

Figure 1 is the example of an arithmetic addition described as the MOD of operation ADD on resource group RG1 = {PC, IMEM, IR, GPR, ADD0}, where PC, IMEM, IR, GPR, and ADD0 are hardware resources. In Fig. 1, the left side description is the MOD of ADD on RG1, and the right side diagram is the corresponding RCG. In Fig. 1, the two pieces of data, *reg1* and *reg2*, from register file GPR at stage 2 are sent to adder ADD0 at stage 3, and then the output is stored in the GPR at stage 5. Signals *rs*, *rt*, and *rd* stand for register indexes from instruction register IR. IMEM is an memory accessing unit that fetches VLIW instructions indicated by the address data from the program counter PC. In this way, the MOD contains not only connection information, but also such high-level architecture information as resource function and purpose.



```
wire [31:0] reg1;
wire [31:0] reg2;
wire [31:0] sum;

stage 1: {
  wire [31:0] cur_pc;
  wire [31:0] inst_word;

  cur_pc    = PC.read();
  inst_word = IMEM.read(cur_pc);
  IR.write(inst_word);
};

stage 2: {
  reg1 = GPR.read0(rs);
  reg2 = GPR.read1(rt);
};

stage 3: {
  sum = ADD0.add(reg1, reg2);
};

stage 4: {
};

stage 5: {
  GPR.write(rd, sum);
};
```

Fig. 1 MOD of ADD on RG1 (ADD, RG1).

3.2 VLIW ASIP Dispatching Model

Describing the dispatching behavior is the main concern of the VLIW ASIP specification. To handle this issue, the dispatching pattern of the VLIW ASIP is modeled as following three concepts: a slot, an operation group, and a resource group. Designers can specify the dispatching rule of the VLIW ASIP by slots, operation groups, resource groups, and their relations.

Slots are parallel dispatching units of the VLIW ASIP. One operation can be dispatched from a slot in one clock cycle.

Let *slot_num* be the number of slots, O be a set of all operations, and $ope_n \in O$ be an operation dispatched from n-th slot, a VLIW instruction *inst* is composed of multiple operations:

$$inst = (ope_1, ope_2, ope_3, ..., ope_{slot\_num}). \tag{2}$$

Note that single-scalar ASIP corresponds to a one slot VLIW ASIP; our method can also deal with both single-scalar and VLIW ASIPs.

Let RG be a set of all resource groups and R is a set of all resources, resource group $rg \in RG$ is a set of hardware resources: $rg \subseteq R$.

The relation between slots and resource groups: RSR is

$$RSR = \{(s, rg) \,|\, 1 \le s \le slot\_num, rg \in RG\}. \tag{3}$$

Pair (s, rg) represents that the operations dispatched from s-th slot are executed on rg.

Let OG be a set of all operation groups, an operation group $og \in OG$ is a set of operations: $og \subseteq O$.

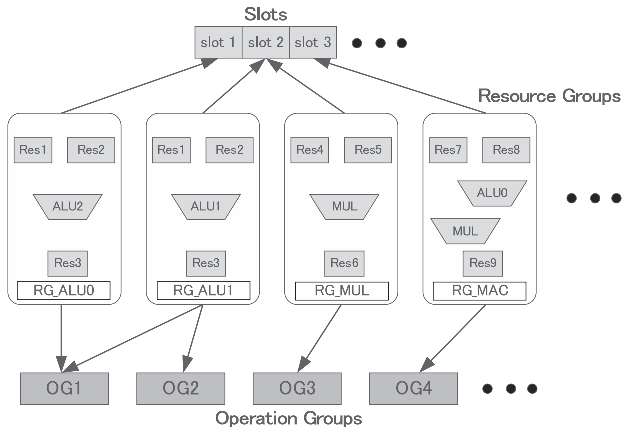


Fig. 2 VLIW ASIP model.

The relation between a resource group and an operation group: RRO is

$$RRO = \{(rg, og) \mid rg \in RG, og \in OG\}. \tag{4}$$

Pair $(rg, og) \in RRO$ indicates that the operations categorized in og can be performed on rg.

As specification, the number of slot $slot\_num$, operations O, resources R, resource groups RG, operation groups OG, the relation between slots and resource groups RSR, and the relation between a resource group and an operation group RRO are given by designers.

Figure 2 illustrates a model of a VLIW ASIP. In Fig. 2, the operations categorized in operation group OG1, which can be performed on resource groups RG_ALU0 and RG_ALU1, can be simultaneously issued from slots 1 and 2.

3.3 VLIW ASIP Controller Model

The decoder model of VLIW ASIPs is shown in **Fig. 3**. To calculate the control logic for data path resources, Dec_{ope} and $Actv_{rg}$ are needed.

Dec_{ope} of operation ope is decode logic on opecode. Dec_{ope} is calculated as

$$Dec_{ope}(inst) = \begin{cases} true & \text{if } \exists ope_n \in inst, \text{code}_{ope} = \text{code}_{ope_n} \\ false & \text{otherwise}, \end{cases} \tag{5}$$

where code_{ope} is the function that returns the opecode of ope and VLIW instruction $inst$ is stored in the instruction register.

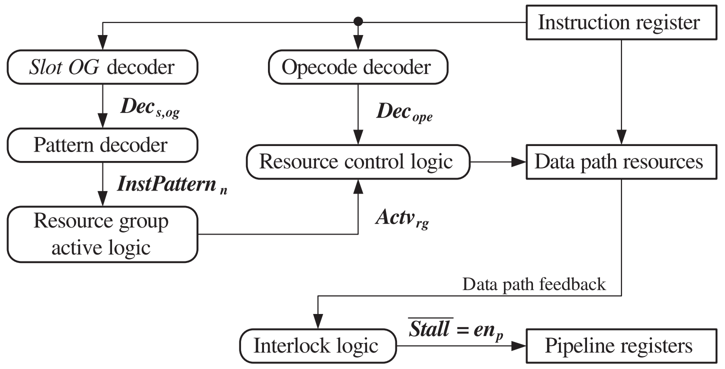


Fig. 3 Decoder model of VLIW ASIPs.

To dispatch operations to appropriate resource groups, the controller calculates $Dec_{s,og}$, $InstPattern_n$, and $Actv_{rg}$. $Dec_{s,og}$ is the decode logic indicating that the operation dispatched from s-th slot belongs to $og \in OG$:

$$Dec_{s,og}(inst) = \bigvee_{\substack{(s,rg) \in RSR \\ ope \in og}} Dec_{ope}(inst) \wedge Exist(rg, ope), \tag{6}$$

$$Exist(rg, ope) = \begin{cases} true & \text{if } \exists (rg, og) \in RRO,\ ope \in og \\ false & \text{otherwise.} \end{cases} \tag{7}$$

In order to activate the appropriate resource groups for the dispatched instruction, pattern detecting logic $InstPattern_n(inst)$ is calculated:

$$InstPattern_n(inst) = \bigwedge_{\substack{T_{IDP_n} \in T_{IDP} \\ (s,rg,og) \in T_{IDP_n}}} Dec_{s,og}(inst), \tag{8}$$

where T_{IDP} is the table of instruction dispatching patterns calculated by the algorithm proposed by Kobayashi, et al. [1),11)], and T_{IDP_n} is the n-th entry in T_{IDP}. Dispatching pattern T_{IDP_n} can be described as a set of (s, rg, og) which indicates that the operations in og can be dispatched from s-th slot and executed on rg.

An example of T_{IDP} is shown in **Table 1**. $InstPattern_n(inst)$ indicates that the pattern of dispatched instruction $inst$. For instance, following Table 1, when $InstPattern_2(inst)$ is true, the pattern of $inst$ is $(OG2, OG2, OG2, OG2)$ and the corresponding resource groups which execute $inst$ are $(RG5, RG2, RG3, RG6)$.

$Actv_{rg}$ is an activation logic of resource group rg:

$$Actv_{rg}(inst) = \bigwedge_{\substack{T_{IDP_n} \in T_{IDP} \\ (s, rg, og) \in T_{IDP_n}}} InstPattern_n(inst). \qquad (9)$$

According to the pattern of VLIW instructions, the assignments of operations to appropriate resource group are controlled by $Actv_{rg}$.

Resource control logic is calculated by using Dec_{ope} and $Actv_{rg}$. On the other hand, the control logic for pipeline registers is calculated only with feed back signals from data path. However, such control incurs unnecessary activation of pipeline registers. As a result, the pipeline registers dissipate unnecessary power.

3.4 VLIW ASIP Generation Flow

VLIW ASIP generation consists of two parts. The first part is data path construction that consists of four procedures: RCG conversion, RCG merging, signal conflict resolution, and pipelining. First, RCGs are converted from MODs. Second, all RCGs are combined by RCG merging to construct a prototype of the data path. Third, multiplexers are inserted, and finally, pipeline registers are inserted into appropriate locations. At each procedure, the generation method retrieves execution conditions that are used to generate steering signals and resource control signals in the next part. The second part is controller construction. The control signals for the resources in the VLIW ASIP are generated using the execution conditions obtained in the previous data path construction part.

Table 1 Example of T_{IDP}.

n	1st slot	2nd slot	3rd slot	4th slot
1	OG1 RG1	OG2 RG2	OG3 RG3	OG4 RG4
2	OG2 RG5	OG2 RG2	OG2 RG3	OG2 RG6
3	OG1 RG1	OG1 RG7	OG3 RG8	OG4 RG4

4. Low-power VLIW ASIP Generation

In this section, a low-power VLIW ASIP generation method is proposed. First, the insertion of gating circuits is discussed, then, the extraction of minimum execution conditions is proposed.

4.1 Insertion of Gating Circuits

Due to the power overhead of the gating circuit, inserting one gating circuit before a few flip flops is ineffective; to increase the clock gating impact, one gating circuit has to be shared by as many as possible. Gate sharing reduces not only the power overhead of the gating circuits but also the power consumption of the clock trees at the same time. In clock tree synthesis, the shared gating circuit can be placed on the upper level of the clock tree. The proposed method also follows this basic insertion strategy. Additionally, as mentioned in Refs. 4) and 12), selecting gating circuit schemes is crucial to increase circuit stability and to decrease implementation overhead. Despite its area and power overhead, the proposed method adopts a flip-flop-based gating scheme because it can block glitch noises that cause incorrect register activation.

4.2 Extraction of Minimum Execution Conditions

In this section, a formal extraction method of minimum execution conditions of the pipeline registers is proposed in the following four data path construction procedures.

4.2.1 RCG Conversion

Resource Connection Graphs (RCGs) are generated from MODs. An RCG is represented by directed graph $G_m = (R_m, E_m)$ where $m = (ope, rg)$ is an identifier of an MOD, R_m is a resources used by m, $E_m = \{(o, i) | o, i \in P\}$ is a set of data transfers used by m, pair (o, i) represents a data transfer from output port o to input port i, and P is a set of all resource ports.

A set of execution conditions $Cond_e$ for each data transfer $e \in E_m$ is retrieved in the RCG extraction. $Cond_e$ is described as

$$Cond_{e \in E_m} = \{(ope, rg) \,|\, m = (ope, rg)\}. \qquad (10)$$

$Cond_e$ denotes that data transfer e is executed when operation ope on resource group rg is dispatched.

Figure 4 shows an example of extracted RCGs. For clarity, the ports of the

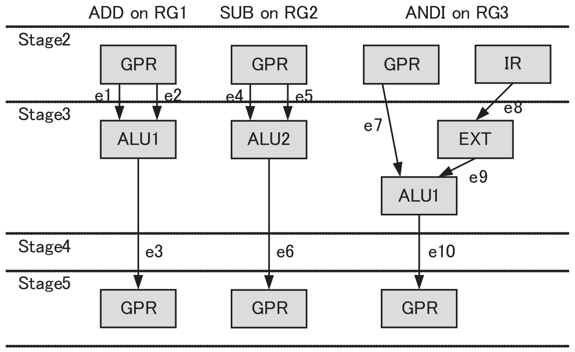


Fig. 4 Converted RCGs.

RCGs and Stage1 are omitted. In the example, the three RCGs correspond to ADD on RG_1, SUB on RG_2, and $ANDI$ on RG_3, respectively. GPR, IR, EXT, $ALU1$, and $ALU2$ are the resources, and the edges labeled $e1$ to $e10$ are the data transfers. Here, sets of the conditions for the data transfers in Fig. 4 are retrieved in the form of Eq. (10) as follows:

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

4.2.2 RCG Merging

After all MODs are converted, they are merged into a unified RCG $G' = (R', E')$. R' and E' are calculated as follows:

$$R' = \bigcup_{\forall m \in M} R_m$$

$$E' = \bigcup_{\forall m \in M} E_m,$$

where M is a set of all identifiers of MODs consisting of the VLIW ASIP. Since data transfers E_m are merged into E', conditions $Cond_e$ of all extracted RCGs are also merged. The new conditions of data transfers $Cond'_e$ are calculated as

$$Cond'_{e' \in E'} = \bigcup_{\forall m \in M, \forall e \in E_m, e'=e} Cond_e. \tag{11}$$

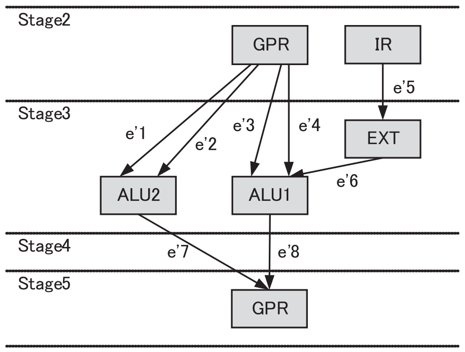


Fig. 5 Unified RCG.

The three operations in the example in Fig. 4 are merged into the unified RCG illustrated as **Fig. 5**. The conditions of 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 the prototype of the data path. Then multiplexers and pipeline registers are inserted in the following procedures.

4.2.3 Signal Conflict Resolution

Multiplexers are inserted in order to resolve signal conflicts occurring in unified RCG G' such as $e'7$ and $e'8$ in Fig. 5. We describe the RCG after the multiplexers are inserted as $G'' = (R'', E'')$.

A multiplexer is inserted before an input port, which is the multiple destination of some data transfers. Here, a set of edges conflicting at identical input port i:

ECI_i is described as

$$ECI_i = \{(o', i') \mid o' \in P,\ i' \in P,\ (o', i') \in E',\ i' = i\}. \quad (12)$$

Since the multiplexers are inserted, the signal connections change. The conditions of data transfers E'' should be calculated, too. A set of the conditions of data transfer $e'' \in E''$ is described as follows:

$$Cond''_{e''} = \begin{cases} \bigcup_{\forall e' \in ECI_{dest_{e''}}} Cond'_{e' \in E'} & \text{if } e'' \in E_{MR}, \\ Cond'_{e' \in E'} & \\ \quad \text{such that } dest_{e'} = dest_{e''} & \text{if } e'' \in E_{RM}, \\ Cond'_{e' \in E'} \text{ such that } e' = e'' & \text{otherwise,} \end{cases} \quad (13)$$

where $dest_e$ is a destination port of 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 selected by a multiplexer.

4.2.4 Pipelining

For pipelining G'', pipeline registers are required for data transfers that cross pipeline stage boundaries. We describe the RCG after the pipeline registers are inserted as G'''. The 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 because one pipeline register is shared by several data transfers from o. A set of edges connected to identical output port o: ECO_o is

$$ECO_o = \{(o'', i'') \mid o'', i'' \in P,\ (o'', i'') \in E'',\ o'' = o\}. \quad (14)$$

In ECO_o, a set of data transfers crossing stage boundaries $ECOX_o$ is

$$ECOX_o = \{(o', i') \mid stage_{o'} < stage_{i'}, \text{for all } (o', i') \in ECO_o\}, \quad (15)$$

where $stage_x$ represents the pipeline stage number to which port x belongs.

In G'', a set of data transfers crossing stage boundaries is

$$E''_{CROSS} = \bigcup_{o \in P_{out}} ECOX_o. \quad (16)$$

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

$$PREG = \bigcup_{(o,i) \in E''_{CROSS}} \{p \mid p = (o, n), stage_o \le n < stage_i\}. \quad (17)$$

Figure 6 depicts the data path after inserting multiplexers MUX and pipeline

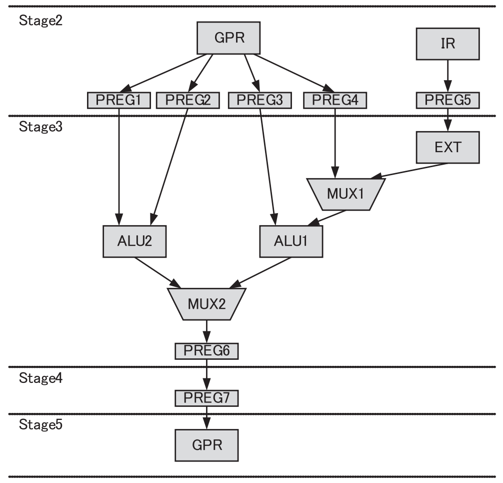


Fig. 6 Constructed data path.

registers PREG in the unified RCG in Fig. 5. Pipeline registers *PREG1* to *PREG7* are inserted in the appropriate points.

Here, the execution conditions of the inserted pipeline registers are calculated. The execution conditions for the pipeline registers are derived from the conditions of the data transfers calculated by Eq. (13). Since the pipeline registers are shared by some data transfers, a set of execution conditions EC_p of pipeline register $p = (o, n)$ is calculated as

$$EC_p = \bigcup_{p=(o,n),\ (o,i) \in ECOX_o} Cond''_{(o,i)}. \quad (18)$$

EC_p is a set of $m = (ope, rg)$ such that ope dispatched to rg requires p for execution. Let $G'''_m \subseteq G'''$ be a necessary data path for executing m and $PREG_m \in G'''_m$ be a set of the pipeline registers that is required for execution by ope dispatched to rg, extracted EC_p can be also described as:

$$EC_p = \{m \mid m \in M, p \in PREG_m\}. \quad (19)$$

For the pipeline registers in Fig. 6, execution conditions EC_p are calculated as follows:

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

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

4.3 Generating Gating Signals with Minimum Execution Conditions

To suppress the unnecessary activations of pipeline register p, we introduce new control signal en'_p with additional logic AL_p:

$$en'_p(inst) = \overline{stall_{stage_p}} \wedge AL_p(inst), \tag{20}$$

$$AL_p(inst) = \bigvee_{\substack{m \in EC_p \\ (ope, rg) = m}} Dec_{ope}(inst) \wedge Actv_{rg}(inst), \tag{21}$$

where EC_p is the derived execution condition in Eq. (19). The modified decoder model of VLIW ASIP with AL_p is shown in **Fig. 7**.

Here, we define correct execution as that the execution result of an operation on the VLIW ASIP generated by the proposed method is same as that by the traditional method.

Theorem 1. *VLIW ASIPs clock-gated with en'_p guarantee correct execution.*

Proof. The new logic en'_p does not prevent the data flow in G'''_m because every pipeline register $p \in PREG_m$ is activated when m is dispatched. Therefore, the execution result of the VLIW ASIP by the proposed method is same as that by the traditional method, i.e., correct execution is guaranteed. □

Theorem 2. *Let a minimum execution condition be an execution condition such that removing one element from the execution condition causes incorrect execution of the generated VLIW ASIP, execution condition Eq. (19) is minimum.*

Proof. Consider that any $m \in EC_p$ is removed, p is not activated when m is dispatched. However, this removing causes incorrect execution because $p \in PREG_m$. Hence, EC_p in Eq. (19) is the minimum execution condition. □

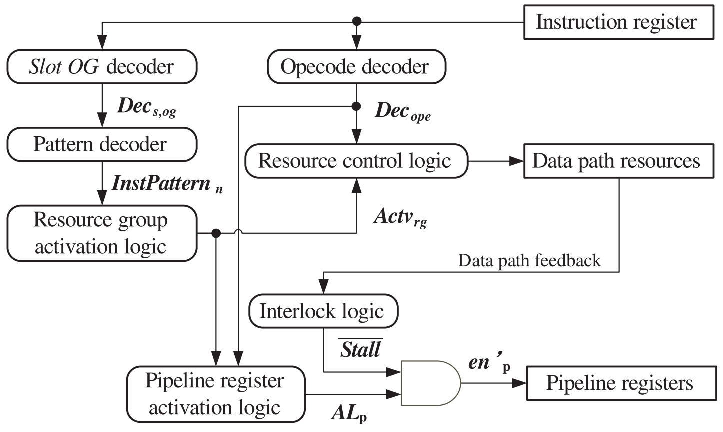


Fig. 7 Modified decoder model.

Using execution condition $EC'_p \supseteq EC_p$ for Eq. (21) can correctly execute operations. Therefore, the traditional VLIW ASIP generation method constantly deals with EC'_p as $EC'_p = M$ for all p. This generation strategy results in reducing area and delay because AL_p can be constantly treated as *true*.

Using Eq. (20), the gating signals of pipeline registers en_{PREG1}, en_{PREG3}, and en_{PREG6} in the VLIW data path illustrated in Fig. 6 can be calculated as follows:

$$
\begin{aligned}
en'_{PREG1}(inst) &= \overline{stall_2} \wedge \{Dec_{SUB}(inst) \wedge Actv_{RG_2}(inst)\},\\
en'_{PREG3}(inst) &= \overline{stall_2} \wedge \{Dec_{ADD}(inst) \wedge Actv_{RG_1}(inst)\\
&\qquad \vee Dec_{ANDI}(inst) \wedge Actv_{RG_3}(inst)\},\\
en'_{PREG6}(inst) &= \overline{stall_3} \wedge \{Dec_{ADD}(inst) \wedge Actv_{RG_1}(inst)\\
&\qquad \vee Dec_{SUB}(inst) \wedge Actv_{RG_2}(inst)\\
&\qquad \vee Dec_{ANDI}(inst) \wedge Actv_{RG_3}(inst)\}.
\end{aligned}
$$

5. Experiments

We carried out two experiments using the integer subset of DLX [13)] to confirm the effectiveness of the proposed method.

For each experiment, we generated the following three types of ASIPs:

NCG: Not clock gated VLIW ASIPs by the traditional generation method

PC: VLIW ASIPs clock gated by Power Compiler

PM: VLIW ASIPs generated by our proposed method.

Note that clock gating was only applied to the pipeline registers in the data path of the generated VLIW ASIPs. The generated ASIPs were synthesized using Design Compiler under a minimizing area constraint and physically synthesized by IC Compiler using a 0.18 μm CMOS technology library operating on 1.8 V.

Programs were randomly generated based on the appearance rate of each instruction. The appearance rates of the instructions in a compiled program are a much more dominant factor than the instruction sequence in the case of using clock gating. Therefore, we modeled the characteristics of the programs as appearance rate in this experiment. For instance, a low instruction per cycle (IPC) program can be modeled as a high NOP appearance rate.

5.1 Evaluation of Hardware Variation

In the first experiment, we generated both single-scalar and VLIW type ASIPs. The single-scalar type ASIPs were extended with Multiply ACcumulate (MAC) instruction by varying the pipeline depth from two to seven stages. The VLIW type ASIPs were designed on a single-scalar ASIP of five stages and homogeneously expanded to two, four, and six slots. Note that the 2-slot processor contains 65 pipeline registers, the 4-slot processor 124, and the 6-slot processor 183; they are large-scale designs. In this experiments, we gave the appearance ratio of the program as follows: load/store are 30%, multiplication is 3.5%, division is 1%, branch/jump are 5%, and integer instruction is 60.5%. These rates are determined by reference to the analysis report of the compiled SPEC benchmarks for MIPS processors[14]. We assumed that all cache access were hit in this experiment.

Note that applying Power Compiler to PM did not affect the circuits because Power Compiler does not insert clock gating into the already gated registers. All pipeline registers of PM are already clock gated.

The experimental results, shown in **Table 2** and **Table 3** reveal the area, delay, and power comparison of the generated ASIPs. Slot # and the pipeline depth are the number of parallel issues and the pipeline depth, respectively. Total power is the power consumption of all circuits, and Pipeline registers stands for the power breakdown by the pipeline registers. Delta is the ratio of the power reduction compared to NCG. Clock tree is the power breakdown of the clock trees, and Skew is the global clock skew of the ASIPs.

Table 2 Power comparison on single-scalar DLX, varying number of pipeline stages.

Pipeline depth	Type	Area	Delay	Total power	Pipeline registers	Delta	Clock tree	Skew
		[μm2]	[ns]	[μW/MHz]	[μW/MHz]	[%]	[μW/MHz]	[ps]
3	NCG	445255	20.09	169.1	17.4		32.8	46
	PC	440625	20.09	147.3	8.5	−51.3	30.7	88
	PM	441330	20.43	146.5	3.1	−78.2	27.8	137
4	NCG	446785	17.34	179.0	25.3		33.3	62
	PC	440632	17.50	163.7	16.1	−36.6	30.6	105
	PM	442388	15.57	153.3	6.6	−73.9	30.4	106
5	NCG	451319	17.33	185.8	31.5		34.9	67
	PC	443782	15.57	169.1	22.8	−27.6	32.2	84
	PM	447188	15.57	155.3	8.2	−73.9	29.8	109
6	NCG	512422	12.10	239.3	63.2		44.0	74
	PC	496831	12.06	209.7	48.4	−23.3	38.6	107
	PM	503537	12.06	174.4	12.1	−80.9	32.4	113
7	NCG	526083	12.10	250.1	68.0		44.9	68
	PC	510137	12.06	218.7	49.2	−27.6	41.4	96
	PM	517711	12.06	183.5	12.4	−81.7	34.6	104

Table 3 Power Comparison on VLIW DLX, varying number of parallel issues.

Slot #	Type	Area	Delay	Total power	Pipeline registers	Delta	Clock tree	Skew
		[μm2]	[ns]	[μW/MHz]	[μW/MHz]	[%]	[μW/MHz]	[ps]
1	NCG	451319	17.33	185.8	31.5		34.9	67
	PC	443782	15.57	169.1	22.8	−27.6	32.2	84
	PM	447188	15.57	155.3	8.2	−73.9	29.8	109
2	NCG	651106	12.06	336.4	112.9		54.1	100
	PC	625762	12.06	298.4	96.7	−14.0	66.9	112
	PM	631341	12.06	222.1	23.7	−79.0	38.3	123
4	NCG	1074753	12.02	514.7	201.2		87.0	117
	PC	1029118	12.07	472.0	197.0	−15.1	81.4	140
	PM	1038249	12.07	318.8	49.8	−88.4	52.3	116
6	NCG	1596043	12.08	703.3	278.0		118.9	78
	PC	1530117	12.12	623.1	234.2	−9.9	110.5	114
	PM	1544843	12.14	403.2	64.8	−76.7	63.9	133

As observed in Table 2, the power consumption of the pipeline registers of PM is reduced approximately 80% compared to NCG in every case, and PC is reduced a maximum of approximately 50%. In addition, the power consumption of every clock tree also decreased. The same trend is observed in Table 3. These results show that the proposed method shuts off more redundant clock supplies than the traditional method.

Area reduction (from NCG to the others) is confirmed because the multiplexers inside the registers are removed when clock gating is applied. Area reduction is an innate advantage of clock gating. On the other hand, negligible area overheads occur owing to the implementation of the minimum execution conditions (from PC to PM). The area overhead reflects the increase of operators in Eq. (20). Compared to Eq. (1), Eq. (20) has more operators and terms to implement minimum execution conditions. Nevertheless, the extra terms in Eq. (20) also exist in the VLIW ASIPs generated by the traditional VLIW generation method, so they do not affect the overheads. The extra operators only result in small area overhead.

On the other hand, critical delay overheads are confirmed to be negligible. Besides being an innate disadvantage of clock gating, clock skew increases in most cases. Though the skew increases for all cases in Tables 2 and 3, the difference of skew between PC and PM is small. The skew results suggest that the proposed method has less extra effect on the clock skew.

For all the ASIPs, the overheads of the calculation time by the proposed method are within several seconds on a workstation operating on 3 GHz using 4 GB memory. This shows that the proposed method has little extra computational overhead compared to the traditional VLIW ASIP generation.

5.2 Evaluation of Software Variation

In the second experiment, we varied the appearance rates of the integer, multi-cycle, load/store, No OPeration (NOP), and control operations on the 4-slot VLIW ASIP to confirm the impact of various programs on the power consumption of the pipeline registers. The multi-cycle operations include multiplication and division, which take 32 cycles to finish operation.

Table 4 shows the results of the five cases. The differences of power reduction are due to the differences of data path utilization in each program. Case 1 in Table 4 corresponds to the result in the previous experiments.

Table 4 Power consumption of pipeline registers in 4-slot VLIW ASIP according to appearance rate.

Case	Int.	Multi Cycle	Load/ store	NOP	Ctrl.	Power [μW/MHz]		
	[%]	[%]	[%]	[%]	[%]	NCG	PC	PM
1	60.5	4.5	30	0	5	201.2	197.0	49.8
2	90	0	5	0	5	264.8	278.1	126.7
3	70	0	5	20	5	246.9	265.9	103.4
4	50	0	5	40	5	241.3	262.1	85.9
5	84	6	5	0	5	190.4	165.9	38.3

With respect to PC, the power consumption in Cases 2 to 4 increased because no multi-cycle operations were issued. In these cases, the power overhead by the gating circuit is just accumulated. The combination of the traditional VLIW generation method and Power Compiler worsens power consumption in such cases. For the same reason, power consumption decreased according to the multi-cycle operations in Case 5.

PM achieves substantial power reduction, as shown in Table 4. In Cases 2, 3, and 4, power consumption decreased, although no multi-cycle operations were issued, showing that gating condition generated by the proposed method stopped unnecessary clock supplies while the pipeline was not stalled. Power consumption decreased while the NOP rate increased because NOP did not activate any data path modules.

6. Conclusion

A low-power VLIW ASIP generation method was proposed in this paper. The proposed method automatically extracts the minimum execution conditions of the pipeline registers in the generated VLIW ASIPs and shuts off the excess clock supplies to the pipeline registers by clock gating. The experimental results showed that the power consumption of the pipeline registers in the VLIW ASIPs generated with the proposed method was reduced about 80% compared to the VLIW ASIPs that were not clock gated, and about 60% compared to the VLIW ASIPs that were clock gated by Power Compiler with negligible delay and area overhead.

Acknowledgments This work is supported by VLSI Design and Education Center (VDEC), The University of Tokyo with the collaboration with Synopsys Corporation.

References

1) Kobayashi, Y., Kobayashi, S., Okuda, K., Sakanushi, K., Takeuchi, Y. and Imai, M.: Synthesizable HDL generation method for configurable VLIW processors, *Proc. ASPDAC*, pp.842–845 (2004).
2) Jacome, M.F., de Veciana, G. and Lapinskii, V.: Exploring performance tradeoffs for clustered VLIW ASIPs, *Proc. ICCAD*, pp.504–510 (2000).
3) Middha, B., Gangwar, A., Kumar, A., Balakrishnan, M. and Ienne, P.: A Trimaran based framework for exploring the design space of VLIW ASIPs with coarse grain functional units, *Proc. ISSS*, pp.2–7 (2002).
4) Lang, T., Musoll, E. and Cortadella, J.: Individual flip-flops with gated clocks for low power datapaths, *IEEE Trans. Circuits Syst. II*, Vol.44, No.6, pp.507–516 (1997).
5) Duarte, D., Vijaykrishnan, N. and Irwin, M.: A clock power model to evaluate impact of architectural and technology optimizations, *IEEE Tran. VLSI*, Vol.10, No.6, pp.844–855 (2002).
6) Benini, L. and Micheli, G.D.: Transformation and synthesis of FSMs for low-power gated-clock implementation, *Proc. ISLPED*, pp.21–26 (1995).
7) Chattopadhyay, A., Geukes, B., Kammler, D., Witte, E.M., Schliebusch, O., Ishebabi, H., Leupers, R., Ascheid, G. and Meyr, H.: Automatic ADL-based operand isolation for embedded processors, *Proc. DATE*, pp.600–605 (2006).
8) Babighian, P., Benini, L. and Macii, E.: A scalable algorithm for RTL insertion of gated clocks based on ODCs computation, *IEEE Trans. Comput.-Aided Des.*, Vol.24, No.1, pp.29–42 (2005).
9) Itoh, M., Takeuchi, Y., Imai, M. and Shiomi, A.: Synthesizable HDL Generation for Pipelined Processors from a Micro-Operation Description, *IEICE Trans. Fundamentals*, Vol.E83-A, No.3, pp.394–400 (2000).
10) Synopsys Inc.: Power Compiler, http://www.synopsys.com/.
11) Kobayashi, Y., Kobayashi, S., Okuda, K., Sakanushi, K., Takeuchi, Y. and Imai, M.: HDL generation method for configurable VLIW processor, *Transactions of Information Processing Society of Japan*, Vol.45, No.5, pp.1311–1321 (2004). (in japanese).
12) Mueller, M., Wortmann, A., Simon, S., Kugel, M. and Schoenauer, T.: The impact of clock gating schemes on the power dissipation of synthesizable register files, *Proc. ISCAS*, pp.II–609–12 Vol.2 (2004).
13) Sailer, P.M. and Kaeli, D.R.: *The DLX instruction set architecture handbook*, Morgan Kaufmann Publishers, Inc., California (1996).
14) Cmelik, R.F., Kong, S.I., Ditzel, D.R. and Kelly, E.J.: An analysis of MIPS and SPARC instruction set utilization on the SPEC benchmarks, *SIGARCH Comput. Archit. News*, Vol.19, No.2, pp.290–302 (1991).

(Received December 1, 2009)
(Revised February 26, 2010)
(Accepted April 14, 2010)
(Released August 16, 2010)

(Recommended by Associate Editor: *Makoto Sugihara*)

Hirofumi Iwato received his B.E. and Master of Information Science and Technology degrees from Osaka University in 2005 and 2007, where he is currently a doctoral candidate. His research interests include design automation, low power design, and VLSI architecture for embedded systems. He is a member of IEEE, IEICE, and IPSJ.

Keishi Sakanushi received his B.E., M.E., and D.E. degrees in electrical and electronics engineering from Tokyo Institute of Technology in 1997, 1999, and 2002. He has been a research associate at Graduate School of Information Science and Technology at Osaka University since the founding of the graduate school in April 2002. His research interests are VLSI layout design, embedded system design, and optimization. He is a member of IEEE, IEICE, and IPSJ.

Yoshinori Takeuchi received his B.E., M.E. and Dr. Eng. degrees from Tokyo Institute of Technology in 1987, 1989 and 1992. From 1992 through 1996, he was a research associate of Department of Engineering at the Tokyo University of Agriculture and Technology. From 1996, he has been at Osaka University. He was a visiting scholar at the University of California, Irvine from 2006 to 2007 and is currently an associate professor of the Graduate School of Information Science and Technology at Osaka University. His research interests include System Level Design, VLSI design, and VLSI CAD. He is a member of IEICE, ACM, and SP, CAS, SSC Society of IEEE, and IPSJ.

Masaharu Imai received his B.S. degree in Electrical Engineering in 1974 and M.S. and Ph.D. degrees in Information Science from Nagoya University in 1976 and 1979. From April 1979 to March 1996, he was with the Department of Information and Computer Sciences, Toyohashi University of Technology, where his final title was professor. He was a visiting professor at the University of South Carolina, Columbia, SC, from 1984 to 1985. Since April 1996, he has been with Osaka University, where he is a professor of Department of Information Systems Engineering, Graduate School of Information Science and Technology. His research interests include ASIP design automation, hardware/software codesign, VLSI architecture, and system level design methodology of embedded systems. Since 1991, he has been working for EDA standardization including VHDL under IEEE and Japan Electronics and Information Technology Industries Association (JEITA). He is a member of IEICE, ACM, and IPSJ.