IPSJ Transactions on System LSI Design Methodology Vol.6 101–111 (Aug. 2013)

[DOI: 10.2197/ipsjtsldm.6.101]

Regular Paper

Energy-efficient High-level Synthesis for HDR Architectures with Clock Gating Based on Concurrency-oriented Scheduling

Hiroyuki Akasaka[1,a) Shin-ya Abe[1] Masao Yanagisawa[2] Nozomu Togawa[1,b)

Received: November 26, 2012, Revised: March 8, 2013,
Accepted: April 26, 2013, Released: August 5, 2013

Abstract: With the miniaturization of LSIs and its increasing performance, demand for high-functional portable devices has grown significantly. At the same time, battery lifetime and device overheating are leading to major design problems hampering further LSI integration. On the other hand, the ratio of an interconnection delay to a gate delay has continued to increase as device feature size decreases. We have to estimate interconnection delays and reduce energy consumption even in a high-level synthesis stage. In this paper, we propose a high-level synthesis algorithm for huddle-based distributed-register architectures (HDR architectures) with clock gatings based on concurrency-oriented scheduling/functional unit binding. We assume coarse-grained clock gatings to huddles and we focus on the number of control steps, or *gating steps*, at which we can apply the clock gating to registers in every huddle. We propose two methods to increase gating steps: One is that we try to schedule and bind operations to be performed at the same timing. By adjusting the clock gating timings in a high-level synthesis stage, we expect that we can enhance the effect of clock gatings more than applying clock gatings after logic synthesis. The other is that we try to synthesize huddles such that each of the synthesized huddles includes registers which have similar or the same clock gating timings. At this time, we determine the clock gating timings to minimize all energy consumption including clock tree energy. The experimental results show that our proposed algorithm reduces energy consumption by a maximum of 23.8% compared with several conventional algorithms.

Keywords: HDR, clock gating, concurrency-oriented scheduling, clock tree, clock gating timing

1. Introduction

With the miniaturization of LSIs and its increasing performance, demand for high-functional portable devices has grown significantly. At the same time, battery lifetime and device overheating are leading to major design problems hampering further LSI integration. On the other hand, the ratio of an interconnection delay to a gate delay has continued to increase as device feature size decreases. We have to estimate interconnection delays and reduce energy consumption even in a high-level synthesis stage.

Clock gating is one of the low-power design techniques in LSI design [4], [8], which reduces the dynamic power of registers by cutting off the clock signal supply when they are not used. Traditionally, clock gating is done at the logic synthesis stage as seen in Ref. [4]. However, since register-writing timing is already determined there, we cannot fully optimize clock gating timing. We should expect better optimization results if clock gating is applied at the high-level synthesis stage combined with floorplanning.

Three types of floorplan-oriented high-level synthesis algorithms have been proposed so far: Cong et al. have proposed a high-level synthesis algorithm based on regular-distributed-register (RDR) architectures [3]. Ohchi et al. have proposed a high-level synthesis algorithm based on generalized-distributed-register (GDR) architectures [5], [6]. In 2012, Abe et al. have introduced the concept of islands into GDR and proposed huddle-based distributed-register (HDR) architectures and its associated high-level synthesis algorithm [1]. In HDR architectures, functional units, registers, and a controller located close to each other are grouped as a *huddle*. The HDR architecture is easy to estimate the interconnection delays and, by giving multiple supply voltages to huddles, it can have power-optimized high-level synthesis. However, the original algorithm in Ref. [1] does not take into account the clock gatings. Moreover, as far as we know, there have been no previous works proposed which deal with clock gatings in floorplan-oriented high-level synthesis.

Based on the previous work described above, we propose a concurrency-oriented scheduling/binding clock gating algorithm for HDR architectures [*1]. This paper is organized as follows: Section 2 first introduces our target architecture and then defines our high-level synthesis problem; Section 3 proposed our high-level synthesis algorithm based on concurrency-oriented scheduling; Section 4 demonstrates experimental results; Section 5 gives several concluding remarks and future works.

[1] Department of Computer Science and Engineering, Waseda University, Shinjuku, Tokyo 169–8555, Japan
[2] Department of Electronic and Photonic Systems, Waseda University, Shinjuku, Tokyo 169–8555, Japan
a) hiroyuki.akasaka@togawa.cs.waseda.ac.jp
b) togawa@togawa.cs.waseda.ac.jp

[*1] Preliminary version of this paper appeared in Ref. [2].

2. Target Architecture and Problem Definition

In this section, we introduce HDR architectures and describe an overview of clock gatings. After that we define our high-level synthesis problem for HDR architectures using clock gatings.

2.1 Huddle-based Distributed-register Architecture

We use an HDR architecture as our target architecture [1]. HDR is an architecture that introduces huddles into GDR and abstracts each module inside an LSI chip. A huddle has a rectangular shape within a range determined by the clock cycle and share functional units, registers, a controller, and level converters in its inside. Since huddles are rectangle but not regular ones unlike RDR, we can achieve small area by packing them effectively. Moreover, huddles abstract modules inside them and then it is easy to add extra modules into each huddle as in RDR.

Figure 1 shows an example of a huddle. The huddle consists of the following components:

- Huddled Local Register (HLR)
  Set of local registers and multiplexers dedicated to each huddle.
- Huddled Functional Unit (HFU)
  Set of functional units collected in each huddle. HFU mainly accesses the HLR in its huddle.
- Finite State Machine (FSM)
  Controller dedicated to each huddle. It controls the HFU and HLR in its huddle.
- Huddled Level Converter (HLC)
  Set of level converters collected in each huddle. It is used to transfer a data to different-voltage huddles

Figure 2 shows an example of the HDR architecture. If we communicate data inside each huddle, data transfer time can be ignored, i.e., it can be done in a single clock cycle. If we communicate data between two huddles, multi-cycle data communication between these huddles can be done as in GDR and RDR. HLC is used when the voltages of the two huddles are different.

2.2 Clock Gating

There are two types of clock gating: One is fine-grained clock gating which cuts off the clock signal to registers one by one. The other is coarse-grained clock gating which cuts off the clock signal to some groups of blocks [9].

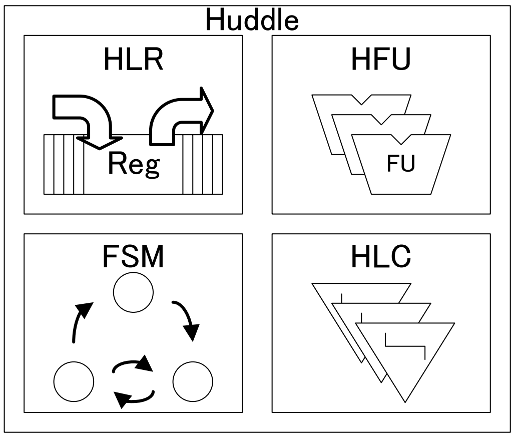


Fig. 1 Huddle.

Fine-grained clock gating has an advantage that each register does not consume extra energy, since we determine the clock gating timing considering the active timing of each register. But fine-grained clock gating must be done at the *leaf side* of a clock tree. Coarse-grained clock gating has a disadvantage that each register may consume extra energy, since we determine the clock gating timing considering the active timing of some register groups. But coarse-grained clock gating can be done at the *root side* of a clock tree. When we focus on a clock tree itself, it consumes energy caused by its drivers and buffers inserted for adjusting the skew. If we apply clock gatings at its root side, low-energy clock tree can be synthesized [7].

In a high-level synthesis stage, the operation execution timing and module floorplanning are not determined yet. This means that we can apply a clock gating at the root side of a clock tree as much as possible during high-level synthesis assuming coarse-grained clock gatings. Coarse-grained clock gating is better for us to use there. Moreover, fine-grained clock gating can be applied even in a logic synthesis stage if necessary.

Now we consider that we apply a clock gating at the root of the entire HDR architecture. However, it is very hard since we have to deal with interconnection delays in clock trees between huddles. One of the reasonable solutions is that we apply a clock gating at the root of each huddle, since we can ignore interconnection delays inside huddles and it is close enough to the root of entire HDR architecture.

Based on the discussion above, we assume coarse-grained clock gatings to huddles (**Fig. 3**), i.e., we set at most one clock gating circuit per huddle. We synthesize huddles such that each of the synthesized huddles include registers having similar or the same clock gating timings [*2].

Figure 3 shows our huddle-based clock tree, which is composed of upper clock trees and lower clock trees. An upper clock tree (bold lines in Fig. 3) is a clock tree from the clock terminal of the chip to huddles and lower clock tree (normal lines in Fig. 3) is a clock tree from a huddle edge to registers. Since we only

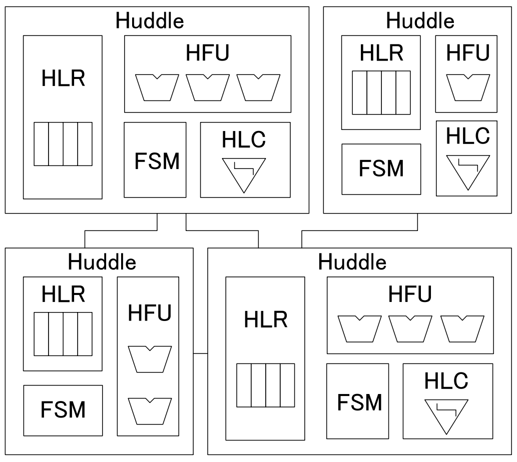


Fig. 2 HDR architecture.

[*2] As can be seen in our experimental results later, energy consumption including clock trees in huddle-based coarse-grained clock gatings is smaller than that in fine-grained clock gatings in most cases.

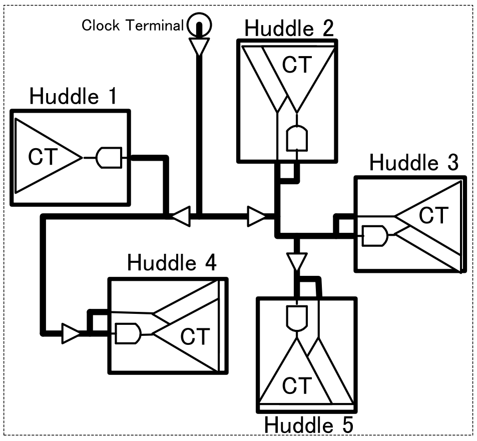


Fig. 3 Our huddle-based clock tree.

consider one clock gating circuit per huddle, lower clock trees in each huddle are composed of at most a gated clock tree and a non-gated clock tree (Huddles 2–5 in Fig. 3). If all the registers in a huddle are clock-gated, i.e., all the register in a huddle have the same gating timings, then its lower clock tree becomes a single gated clock tree as in Huddle 1 in Fig. 3. We prepare a clock tree for each of the supply voltages.

2.3 Problem Formulation

An input behavioral description is represented by a CDFG (Control-Data Flow Graph). T_{clk} refers to a clock period constraint and $S_{\max}$ refers to a latency constraint. $S_{\max}$ shows the maximum allowable number of control steps to execute a given CDFG.

Let $F = \{f_1, \cdots, f_p\}$ be a set of p functional units and $D_f(f_i)$ be a delay of the functional unit f_i in F. $S_f(f_i)$ shows the number of control steps required to execute the functional unit f_i and $S_f(f_i)$ is defined by $S_f(f_i) = \lceil D_f(f_i)/T_{clk} \rceil$. Let $E(f_i)$ be the energy consumed by the functional unit f_i.

Let $H = \{h_1, \cdots, h_q\}$ be a set of q huddles ($q \leq p$). $Hud(f_i)$ is the huddle to which the functional unit f_i is bound. $F(h_j)$ is a set of functional units which are bound to the huddle h_j. Let $R(h_j)$ be a set of registers which are bound to the huddle h_j and $D_{reg}(h_j)$ be a delay of HLR in the huddle h_j.

$Slack(f_i)$ is defined by:

$$Slack(f_i) = T_{clk} \cdot S_f(f_i) - D_f(f_i) \tag{1}$$

$Slack(f_i)$ shows the slack time which can be used by data transfer for f_i's succeeding operations.

The width and height of each huddle must satisfy the following *huddle size constraint*:

$$2 \cdot D_w(W(h_j) + H(h_j)) + D_{reg}(h_j) \leq \min_{f_i \in F(h_j)} \{Slack(f_i)\} \tag{2}$$

where $W(h_j)$ and $H(h_j)$ are the width and height of the huddle h_j, respectively. $D_w(x)$ is an interconnection delay when the wiring length is x. $D_w(x)$ is proportional to the square of x and defined by $D_w(x) = C_d x^2$, where C_d is the interconnection delay coefficient.

Let $Dist(h_j, h_k)$ be the Manhattan distance between the huddles h_j and h_k. Then $D_w(Dist(h_j, h_k))$ shows the interconnection delay between them. Let $V(h_j)$ and $V(h_k)$ be supply voltages to h_j and h_k and $D_{lc}(V(h_j), V(h_k))$ be the level convert delay from h_j to h_k. Let f_i be a functional unit bound to the huddle h_j, i.e., $Hud(f_i) = h_j$. $Tr(f_i, h_k)$ shows the inter-huddle data transfer delay from f_i to HLR in the huddle h_k which is defined by:

$$\begin{aligned} Tr(f_i, h_k) &= D_w(Dist(h_j, h_k)) \\ &\quad + D_{lc}(V(h_j), V(h_k)) + D_{reg}(h_k) \end{aligned} \tag{3}$$

$DT(f_i, h_k)$ shows the number of clock cycles required to transfer data from f_i to h_k which is defined by:

$$DT(f_i, h_k) = \begin{cases} 0 & (Slack(f_i) \geq Tr(f_i, h_k)) \\ \lceil Tr(f_i, h_k)/T_{clk} \rceil & (Slack(f_i) < Tr(f_i, h_k)) \end{cases} \tag{4}$$

Based on the above definitions, our high-level synthesis problem is defined as follows:

Definition 1. *Our high-level synthesis problem is, for a given CDFG, a clock period constraint, a latency constraint, and a set of functional units, to assign each operation node to a control step and a functional unit, to bind each functional unit to each huddle, to assign a supply voltage to each huddle, to apply a clock gating to each huddle so that the given CDFG is executed correctly considering multi-cycle interconnect communications as in Eq.* (*4*)*. The objective is to minimize the total energy consumption.*

3. Energy-efficient High-level Synthesis Algorithm for HDR Architectures with Clock Gatings

A high-level synthesis algorithm based on HDR architectures utilizing multiple supply voltages was proposed in Ref. [1]. It is very effective in terms of low energy optimization since it achieves 25.8% energy reduction, but it does not deal with clock gatings. We can expect that we have more energy savings if we can incorporate clock gatings into Ref. [1]. Then we will extend the original algorithm so that it can effectively utilizes clock gatings.

To realize a low energy high-level synthesis by applying clock gatings to an HDR architecture, it is necessary to increase *gating steps* in each huddle, i.e., the number of control steps to cut off the clock signal to registers in each huddle without increasing extra functional units nor extra registers. As increasing the number of gating steps, we can reduce the energy consumption by applying a clock gating to each huddle.

The algorithm [1] is composed of the five processes below and they are repeated until no further improvement is seen. When we consider increasing the number of gating steps, we can also have the five options, i.e., we will deal with clock gatings at each of the five processes below:

Option 1: Initial huddling
Option 2: Scheduling/FU (Functional unit) binding
Option 3: Register binding/Controller synthesis/Floorplanning
Option 4: Huddling
Option 5: Unhuddling

In Option 1, we try to start our high-level synthesis by initially grouping the functional units having the same or similar gating timings into a huddle and then we can increase the gating steps. However, we cannot identify functional units to assign to the same huddle because no timing information is available here. It is impossible for us to apply Option 1 to Ref. [1].

In Option 2, we try to schedule as many operations as possible to the same control steps and thus make the "empty" control steps. This leads to increasing the gating steps. Option 2 is one of the most important options to increase the gating steps.

In Option 3, we try to minimize the number of registers to achieve low energy consumption. In Ref. [1], it is already realized.

In Option 4, we try to merge several huddles used at the same or similar timings into a single huddle. Since we already have the timing information in the "huddling" step, we can easily identify huddles used at the same or similar timing. Option 4 is also one of the most important options to increase the gating steps.

In Option 5, we try to partition the functional units in each huddle used at the different timings into different huddles. In this option, gating steps may increase but we may have extra unnecessary huddles. Option 5 is not a good option in this sense.

Based on the above discussions, Option 2 and Option 4 are the two important options. In Option 2, we schedule as many operations as possible to the same control steps and thus make the "empty" control steps (which is called "Concurrency-oriented scheduling / FU binding"). In Option 4, it is important to identify huddles used at the same or similar timings, i.e., we first have to determine the clock gating timings to minimize all energy consumption including clock tree energy (which will be done in "CG timing calculation considering CT" before the huddling step). After that, we will merge the huddles with the same or similar gating timing into a single huddle at the huddling step (which is called "CG huddling").

Figure 4 shows our proposed algorithm: "Initial huddling" prepares a huddle for each input functional unit; "Concurrency-oriented Scheduling/FU binding" determines the operation timings and assigns operations to functional units so that the number of gating steps is increased; "Register binding" assigns variables to registers; "Controller synthesis" synthesizes each controller in a huddle and "floorplanning" performs huddle floorplanning;

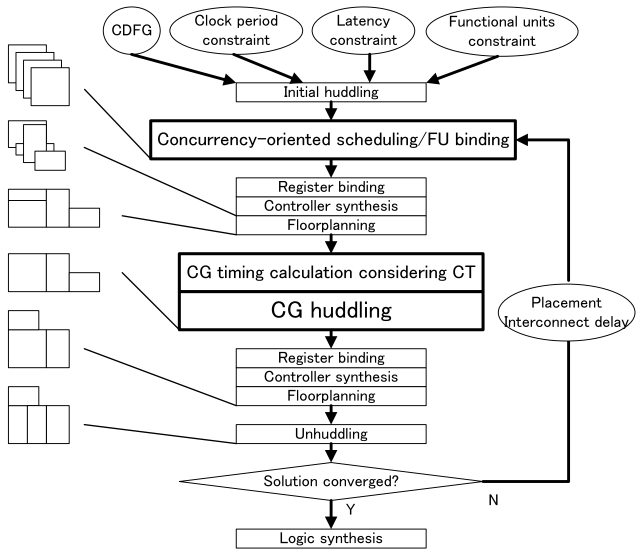


Fig. 4 Proposed algorithm.

"CG timing calculation considering CT" calculates the clock gating timings of huddles considering clock tree energy and "CG huddling" merges several huddles into a single huddle considering the gating timings; "Unhuddling" partitions a single huddle into two or more huddles in order to satisfy the given timing constrains. By repeatedly performing these steps, we finally have an energy-saving high-level synthesis result with floorplanning.

We can use the exactly the same algorithms as Ref. [1] other than "Concurrency-oriented Scheduling/FU binding," "CG timing calculation considering CT," and "CG huddling." Then we propose here "Concurrency-oriented Scheduling/FU binding," "CG timing calculation considering CT," and "CG huddling."

3.1 Concurrency-oriented Scheduling/FU Binding

In "Concurrency-oriented Scheduling/FU binding," we use CDFG, a clock cycle constraint, a latency constraint, and a set of functional units as input. We can also use information of placement and interconnection delays obtained in the previous iteration since we employ the iterative improvement flow as in Fig. 4. In this step, we will assign each operation node to a control step, bind it to a functional unit, and determine a supply voltage of each huddle.

In order to increase control steps at which we can apply a clock gating to registers, the active timings of registers in each huddle should be aligned somehow, where active timings of registers mean the timings at which operation or register-to-register communication between huddles are performed.

Since the original algorithm in Ref. [1] effectively realizes low-energy scheduling/FU binding utilizing multiple-supply voltages, our proposed algorithm here is also based on it and extend it so that it can align the active timings of registers in each huddle. Given a scheduling/FU binding result obtained by the original scheduling/FU binding algorithm in Ref. [1], we can have the four strategies to align the active timings of registers:

Strategy 1: Try to re-assign an operation to an earlier control step

Strategy 2: Try to re-assign a register-to-register communication to an earlier control step

Strategy 3: Try to re-assign an operation to a later control step

Strategy 4: Try to re-assign a register-to-register communication to a later control step

Strategies 1 and 2 re-assign some of operations and register-to-register communications to earlier control steps. However, the original algorithm in Ref. [1] is based on list scheduling which tries to assign operations to the earliest possible control steps without violating their execution order and available functional units. If we incorporate Strategies 1 and 2 into Ref. [1], we can easily violate operation execution order and/or will exceed available functional units/registers. In that sense, we can conclude that Strategies 1 and 2 are not good choices.

Strategy 3 re-assigns an operation to a later control step. If it still satisfies the operation execution order and will not exceed available functional units, it is possible to align active timings of registers there.

Similarly, Strategy 4 re-assigns a register-to-register communication to a later control step. If it still satisfies the operation execution order and will not exceed available registers, it is also possible to align active timings of registers there.

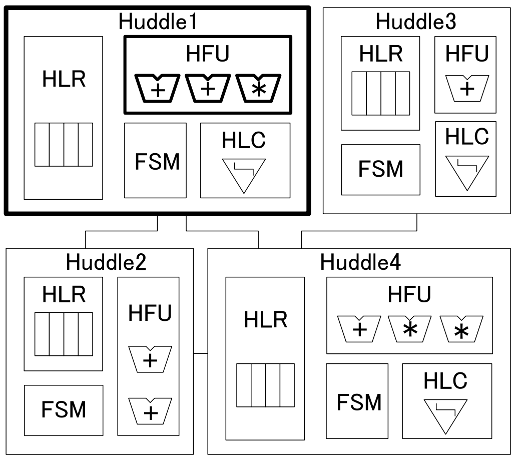
Strategies 3 and 4 must be good choices to align active timings of registers in a huddle.

Example 1. ***Figure 5*** *shows the example of aligning the active timings of registers in Strategy 3. In this example, we assume that the huddle 1 has one multiplier and two adders and any operation can be performed in a single control step. We also assume that functional units and registers have the same bit width. As discussed in Section 2.2, we consider course-grained clock gatings to huddles, i.e., we set at most one clock gating circuit per huddle.*

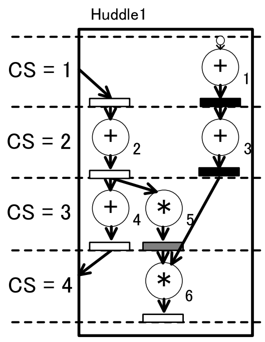
In Fig. 5, we first obtain a scheduling and FU binding result by applying the original scheduling and FU binding algorithm in Ref. [1] (*Fig. 5* (*b*)). *In Fig. 5* (*b*), *we show the operation scheduling/binding and when the register writings occur. In Fig. 5* (*d*), *a check mark "✓" shows a register writing timing. If we apply a clock gating to Register C, we can cut off the clock signal to it at Steps 1, 2 and 4. A circle mark "○" in Fig. 5* (*d*) *shows each gating step. But we cannot apply further clock gatings to Registers A nor B since their active timings are not aligned with Register C and we can have at most one clock gating circuit per huddle. In this case, we totally have three gating steps* (*Steps 1, 2 and 4*).

Then, we re-assign the node 3 from Step 2 ($CS = 2$) *to Step 3* ($CS = 3$) *as in* (*Fig. 5* (*c*)). *If we apply clock gatings to Register B and C, we can cut off the clock signal to them at Steps 2 and 4* (*Fig. 5* (*e*)). *Though we cannot apply a clock gating to Register A, we can totally have four gating steps in this case* (*Steps 2 and 4 for Register B and Steps 2 and 4 for Register C*). *This is due to aligning the active timings of Registers B and C by moving the node 3 from Step 2 to Step 3. We can expect that the total energy consumption will be reduced by applying clock gatings to them.*

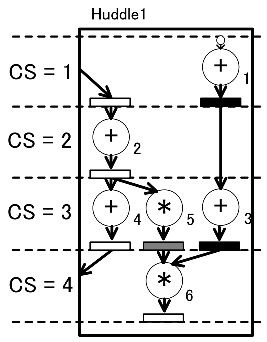
Example 2. ***Figure 6*** *shows the example of aligning the active timings of registers in Strategy 4. We consider the same assump-*



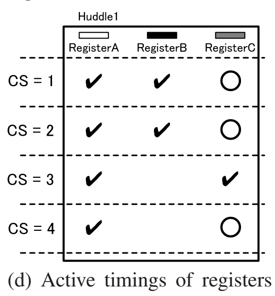
(a) HDR architecture in Strategy 3.



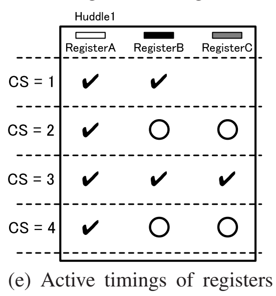
(b) A result obtained by the original scheduling/ FU binding [1].



(c) A result obtained by our proposed concurrency-oriented scheduling/ FU binding.

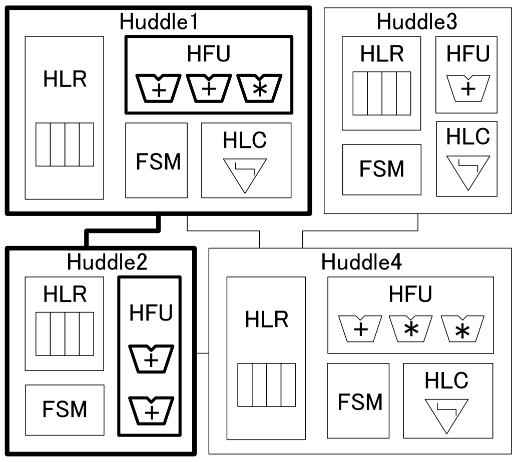


(d) Active timings of registers for (b).

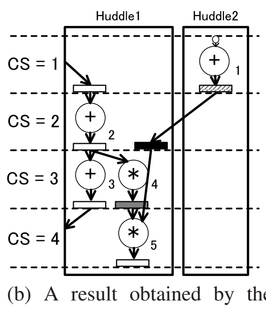


(e) Active timings of registers for (c).

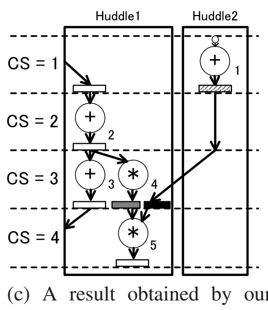
Fig. 5 Aligning active timings of registers in Strategy 3. The check marks "✓" show register writing timings and the circle marks "○" show gating steps where we can cut off the clock signal.



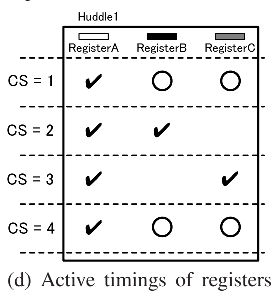
(a) HDR architecture in Strategy 4.



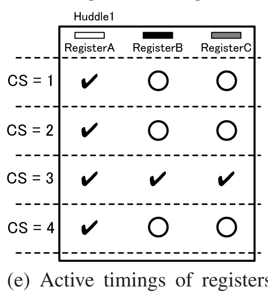
(b) A result obtained by the original scheduling/FU binding [1].



(c) A result obtained by our proposed concurrency-oriented scheduling/ FU binding.



(d) Active timings of registers for (b) in Huddle 1.



(e) Active timings of registers for (c) in Huddle 1.

Fig. 6 Aligning active timings of registers in Strategy 4. The check marks "✓" show register writing timings and the circle marks "○" show gating steps where we can cut off the clock signal.

tions as the previous example.

In Fig. 6, we first obtain a scheduling and FU binding result by applying the original scheduling and FU binding algorithm in Ref. [*1*] *(Fig. 6* (*b*)). *Now let us look at registers in Huddle 1. In Fig. 6* (*d*)*, a check mark "✓" shows a register writing timing in Huddle 1. If we apply clock gatings to Registers B and C, we can cut off the clock signal to it at Steps 1 and 4. A circle mark "∘" in Fig. 6* (*d*) *shows each gating step. We cannot apply further clock gatings to Register A since its active timings are not aligned and we can have at most one clock gating circuit per huddle. In this case, we totally have four gating steps* (*Steps 1 and 4 for Register B and Steps 1 and 4 for Register C*)*.*

Then, we re-assign the register-to-register communication between Huddle 2 and Huddle 1 from Step 2 ($CS = 2$) *to Step 3* ($CS = 3$) *as in* (*Fig. 6* (*c*))*. If we apply clock gatings to Registers B and C in Huddle 1, we can cut off the clock signal to them at Steps 1, 2 and 4* (*Fig. 6* (*e*))*. Though we cannot apply a clock gating to Register A, we can totally have six gating steps in this case* (*Steps 1, 2 and 4 for Register B and Steps 1, 2 and 4 for Register C*)*. This is due to aligning the active timings of the register-to-register communication by moving it from Step 2 to Step 3. We can expect that the total energy consumption will be reduced by applying clock gatings to registers in Huddle 1.*

Based on the above discussions, we will extend the original scheduling/FU binding [1] and propose "Concurrency-oriented Scheduling/FU binding" incorporating Strategies 3 and 4, where we move operations and register-to-register communications to later control steps so as to align the active timings of registers in huddles.

Let o be an operation or a register-to-register communication in a scheduling/FU binding result obtained by the original scheduling/FU binding algorithm [1]. Let $CS_S(o)$ and $CS_L(o)$ be the earliest control step and the latest control step at which o can be assigned without exceeding required functional units nor registers, respectively. Then the mobility $Mob(o)$ is defined by:

$$Mob(o) = CS_L(o) - CS_S(o) \tag{5}$$

Our "Concurrency-oriented Scheduling/FU binding" algorithm is summarized as follows:

```
Initial phase:
  We perform the original scheduling/FU binding [1].
Improvement phase:
  (1) Let $cs \leftarrow (S_{\max} - 1)$, where $S_{\max}$ shows the maximum control step given as input.
  (2) While ($cs > 0$), perform the following loop (a)–(b):
      (a) Let $o$ be an operation or a register-to-register communication assigned to $cs$. Let $hud(o)$ be the huddle to which $o$ is assigned. Perform the following processes to $o$ in the ascending order of $Mob(o)$:
          (i) Try to move $o$ to each later control step $i$ ($cs \leq i \leq S_{\max}$) within the mobility range without changing its FU binding and exceeding the required registers, and count how many register writings totally occur at $i$. Let $AllAct(i)$ ($cs \leq i \leq S_{\max}$) be such total number of register writings at $i$ in $hud(o)$.
          (ii) Move $o$ to the control step $i_{\max}$ giving the maximum $AllAct(i)$.
      (b) If we have tried all the operations and register-to-register communications assigned to $cs$, $cs \leftarrow cs - 1$ and go back to (a). If not, go to (i) and try the next operation or register-to-register communication.
```

By using our proposed algorithm, we can align as many operations and register-to-register communications as possible and then we can increase clock cut-off steps. Finally we can expect that we effectively apply clock gatings to each huddle and have a low-energy scheduling/FU binding.

3.2 CG Timing Calculation Considering CT

"CG timing calculation considering CT" assigns clock gating timings to each huddle based on the active timings of registers and the clock tree energy. Now let focus on a huddle h in our HDR architecture.

Let $Act(r_j, i)$ be the active timing information of register r_j in h at the step i. $Act(r_j, i)$ is a 0-1 variable and $Act(r_j, i) = 1$ when the register r_j is used at the step i. Otherwise, $Act(r_j, i) = 0$. Let R be a subset of registers in the huddle h and $Act(R, i)$ shows the active timing information of $R = \{r_1, \cdots, r_k\}$ of the k registers at the step i which is defined by:

$$Act(R, i) = \bigcup_{r_j \in R} Act(r_j, i) \tag{6}$$

$CG(R, i)$ shows the *clock gating timing* of the subset R of the k registers at the step i which is defined by:

$$CG(R, i) = \overline{Act(R, i)} \tag{7}$$

Figure 7 shows an example of $Act(r_j, i)$ and $CG(R, i)$ for various patterns of the subset R in the huddle.

Using the latency constraint $S_{\max}$, the gating step count $Step(R)$ can be defined by:

$$Step(R) = k \times \sum_{i=1}^{S_{\max}} CG(R, i) \tag{8}$$

where $k = |R|$.

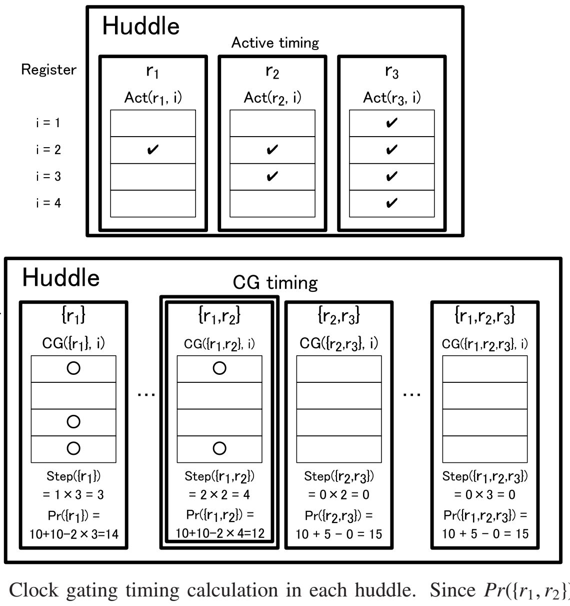


Fig. 7 Clock gating timing calculation in each huddle. Since $Pr(\{r_1, r_2\})$ gives the minimum, we will apply clock gatings to both r_1 and r_2 in this huddle. We will cut off the clock signal to r_1 and r_2 at Steps 1 and 4 and will supply the clock signal to r_3 at all the steps.

Let E_{step} be the energy consumption of a register in one step [*3]. As mentioned in Section 2.2, an upper clock tree (bold lines in Fig. 3) is a clock tree from the clock terminal of the chip to huddles and lower clock tree (normal lines in Fig. 3) is a clock tree from a huddle edge to registers. Let $E_u(R)$ be the energy consumption of the upper clock tree for the huddle h when we apply a clock gating to the register subset R in h. Let $E_d(R)$ be the energy consumption of the lower clock trees in h when we apply a clock gating to the register subset R in h. Then the cost $Pr(R)$ can be defined by:

$$Pr(R) = (E_u(R) + E_d(R)) - E_{step} \times Step(R), \qquad (9)$$

where $E_u(R)$ and $E_d(R)$ can be obtained by using the equations in Ref. [12]. The second term ($E_{step} \times Step(R)$) in Eq. (9) directly shows how much energy consumption can be reduced by cutting off the clock signal to R. We need the first term ($E_u(R) + E_d(R)$) because of the following reason: The configuration of lower clock trees in h must be changed when the subset R of clock-gating registers changes. $E_d(R)$ is directly dependent on the lower clock tree configuration in h and calculated mainly based on how many lower clock trees are required inside h and how many sinks each lower clock tree requires. $E_u(R)$ is calculated mainly based on the number of upper clock sinks and their positions, which are dependent on lower clock tree configurations. In each huddle other than h, we use a lower clock tree configuration determined in the previous iteration or just before at this iteration to calculate $E_u(R)$. In h, we use the lower clock tree configuration when we apply a clock gating to the subset R. Since lower clock trees in h as well as its upper clock tree can be changed when the subset R of clock-gating registers changes in h, we have to evaluate not only the second term but also the first term as in Eq. (9).

Let $R(h)$ be the set of all the registers in the huddle h. For every subset R in $R(h)$, we can calculate $Pr(R)$ and find out the one which gives the minimum $Pr(R)$ value. Let $R_{\min}$ be such subset in $R(h)$ and $Pr(R_{\min})$ shows the minimum value. When we cut off the clock signal to each register in $R_{\min}$ at the steps i satisfying $CG(R_{\min}, i) = 1$, it will lead to the lowest energy consumption by applying the clock gatings to the huddle h.

Example 3. *In Fig. 7, we have three registers r_1, r_2 and r_3 in the huddle. The register writing timings and $Act(r_j, i)$ for every register r_i and Step i are also given. The check mark "✓" shows that $Act(r_j, i) = 1$. Since we have three registers, we can have seven register subset patterns of $\{r_1\}, \cdots, \{r_1, r_2\}, \{r_2, r_3\}, \cdots, \{r_1, r_2, r_3\}$.*

Then we can calculate $Act(R, i)$ and $CG(R, i)$ for each subset R and Step i. Figure 7 also shows $CG(\{r_1\}, i), \cdots, CG(\{r_1, r_2\}, i), CG(\{r_2, r_3\}, i), \cdots, CG(\{r_1, r_2, r_3\}, i)$. The circle mark "○" shows that $CG(R, i) = 1$.

Assume that $E_{step} = 2$ and $E_u(R) = 10$ for every subset R. $E_d(R) = 10$ when we use both a gated lower clock tree and a non-gated lower clock tree in the huddle. $E_d(R) = 5$ when we use only a non-gated lower clock tree in the huddle. We can calculate gating steps and, by using them, we can have $Pr(\{r_1\}), \cdots, Pr(\{r_1, r_2\}), Pr(\{r_2, r_3\}), \cdots, Pr(\{r_1, r_2, r_3\})$. Note that, when we pick up $\{r_2, r_3\}$ or $\{r_1, r_2, r_3\}$ for gating register candidates, we have no gating steps. In this case, we use only a non-gated lower clock tree in the huddle (in this case $E_d(\{r_2, r_3\}) = E_d(\{r_1, r_2, r_3\}) = 5$).

Finally we have that $Pr(\{r_1, r_2\})$ gives the minimum and will cut off the clock signal to r_1 and r_2 at Steps 1 and 4 in this huddle using a clock gating circuit.

[*3] Here we assume that every register has the same bit width for simplicity and then we can define E_{Step} to be a single value. The similar discussions can be applied to registers with the different bit widths by assuming a different E_{Step} value for every register.

3.3 CG Huddling

In "CG huddling," we merge several huddles into a single huddle. We first calculate the merge priorities for huddles and merge them in the descending order of them.

Merge Priority

To calculate the merge priority, we propose *similarity* in addition to adjacency and the number of connections proposed in Ref. [1]. Similarity represents how close the clock gating timings of two huddles are. By merging the two huddles used at the same or similar timing, it is possible to cut off the clock signal to the registers in the merged huddle at as many steps as possible.

We propose the similarity $Sim(h_j, h_k)$ between the huddles h_j and h_k as follows:

$$Sim(h_j, h_k) = S_{\max} - \sum_{i=1}^{S_{\max}} CG(h_j, i) \oplus CG(h_k, i) \qquad (10)$$

where $S_{\max}$ is the latency constraint. $CG(h_j, i)$ and $CG(h_k, i)$ are the clock gating timings of h_j and h_k calculated in the "CG timing calculation," respectively. $Sim(h_j, h_k)$ shows the number of control steps where we can cut off the clock signal if we merge h_j and h_k. **Figure 8** shows an example of the similarity between the two huddles.

The merge priority $Pr(h_j, h_k)$ between two huddles h_j and h_k is set to be:

$$Pr(h_j, h_k) = Con(h_j, h_k) \times Adj(h_j, h_k) \times Sim(h_j, h_k), \qquad (11)$$

where $Con(h_j, h_k)$ and $Adj(h_j, h_k)$ shows the number of connections and adjacency between the two huddles, respectively [1].

Based on the merge priority, we will merge two huddles h_j and h_k into a single huddle according to the original algorithm of Ref. [1] and apply a clock gating to the steps where we can cut off the clock signal to both h_j and h_k.

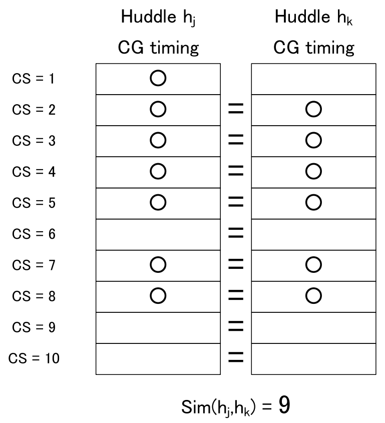


Fig. 8 Similarity. The circle mark "○" shows the gating step.

4. Experimental Results

4.1 Setup

We have implemented our algorithm in C++ and performed experimental evaluations. We used AMD Opteron 2360SE 2.5 GHz × 2 PC with 16 GB memory. We applied our algorithm to hal (11 nodes), parker (22 nodes), dct (48 nodes), jacobi (48 nodes including condtional branches), fir filter (75 nodes), ewf3 (102 nodes), and copy (378 nodes including conditional branches) [1], [5], [6], [10].

In the experiments, functional units and registers are assumed to have 16-bit width under the 90 nm technology. The clock period constraint is given to be 2.5 ns. Each latency constraint is just given to a round number as seen in Ref. [1]. Numbers of FUs are given so that we can satisfy the given latency constraint with the almost minimum possible FUs. The interconnection delays are assumed to be proportional to square of the wiring length and set interconnection delays to be 1ns when the wiring length is 250 μm [1]. **Tables 1** and **2** show our functional unit and register specifications. Supply voltages are assumed to be 0.8 V, 1.0 V, and 1.2 V. We used Synopsys Design Compiler to obtain energy and power for HDR architecture components. We consider a clock tree for each of three supply voltages and its energy is obtained by using the equations in Ref. [12].

Energy consumption in each application program is obtained based on the energy consumptions of (a) FUs (including their multiplexers), (b) level converters, (c) registers (including their multiplexers), (d) controllers, and (e) clock trees. Our energy consumption value is then obtained by summing up those of (a)–(e) as seen in Refs. [1], [11], where we assume that the components which are prepared in advance can be used directly in high-level synthesis. We obtained the energy consumptions of (a)–(d) by using Design Compiler. They may include some errors compared to actual values, but which are strongly dependent on Design Compiler. Since clock tree layout is not fixed in a high-level synthesis stage, we just cannot know the actual energy consumption in this stage. We then use the equations in Ref. [12] to calculate the energy consumption of (e) clock trees in our experiments. Clock tree energy estimation in Ref. [12] may have some amount of errors compared to an actual value but, throughout this experimental evaluation, we obtained the energy consumption in each application program as discussed above for our proposed algorithm as well as for conventional algorithms, which means that our comparisons can be quite fair. In a similar way, we obtained the delays of (a)–(d), where we just ignore the delays of (e) clock trees.

Table 1 Functional unit specification.

Functional unit	Area [μm^2]	Delay [ns]	Energy [pJ]	Leak power [μW]
Adder (1.2 V)	386	0.75	0.092	3.9
Adder (1.0 V)	386	1.22	0.064	3.2
Adder (0.8 V)	386	2.71	0.041	2.6
Substractor (1.2 V)	417	0.78	0.097	4.2
Substractor (1.0 V)	417	1.27	0.067	3.5
Substractor (0.8 V)	417	2.82	0.043	2.8
Multiplier (1.2 V)	2,161	1.65	1.135	19.8
Multiplier (1.0 V)	2,161	2.7	0.788	16.5
Multiplier (0.8 V)	2,161	6.0	0.504	13.2
Divider (1.2 V)	6,404	5.91	1.865	670.8
Divider (1.0 V)	6,404	9.66	1.295	559.0
Divider (0.8 V)	6,404	21.47	0.829	447.2
Comparator (1.2 V)	116	0.51	0.017	0.80
Comparator (1.0 V)	116	0.83	0.012	0.67
Comparator (0.8 V)	116	1.84	0.008	0.54
Shifter (1.2 V)	294	0.54	0.075	2.5
Shifter (1.0 V)	294	0.89	0.052	2.1
Shifter (0.8 V)	294	1.98	0.033	1.7

Table 2 Register specification.

	Clock gating	Area [μm^2]	Energy [pJ]	Leak power [μW]
Register (1.2 V)	Yes	272	0.546	0.0018
	No	272	0.743	0.0017
Register (1.0 V)	Yes	272	0.379	0.0015
	No	272	0.516	0.0014
Register (0.8 V)	Yes	272	0.243	0.0012
	No	272	0.330	0.0011

We have compared our algorithm with the following algorithms:

(1) The original algorithm [1],
(2) The original algorithm [1] followed by the coarse-grained clock gating (Ref. [1]+CGCG),
(3) The original algorithm [1] followed by the fine-grained clock gating (Ref. [1]+FGCG), and
(4) Our algorithm without using concurrency-oriented scheduling/FU binding but using the original scheduling/FU binding (Ours w/o Section 3.1).

In (2) and (4) above, we applied huddle-based coarse-grained clock gating whereas we applied fine-grained clock gating to (3). In (2), we used the algorithm described in Section 3.2 for coarse-grained clock gating. In (3), we constructed a lower clock tree to every set of registers with the same active timings in each huddle, which controls clock gating to each register. We may construct too many lower clock trees for some huddle in this case.

4.2 Energy Evaluation

Table 3 shows our experimental results where CT energy refers to clock tree energy including upper clock trees and lower clock trees. All energy consumption includes energy for all the components in HDR as well as clock tree energy. Overall, all energy consumption of our proposed algorithm achieves the minimum in all the cases and is reduced by a maximum of 23.8% and an average of 14.4% compared with (1) the original algorithm [1].

Particularly, clock tree energy for fine-grained clock gating is approximately 2–6 times more than that for course-grained clock gating, although dynamic energy in registers for fine-grained clock gating is the minimum in all the cases. This indicates that high-level synthesis with huddle-based course-grained clock gating is very effective in terms of total energy reduction.

CPU time is largely dependent on the number of iterations and there are almost no significant differences in the number of iterations in each application program, through the number of iterations may increase in some case (for example, Ours w/o Section 3.1 of dct). This is because all the algorithms used in our experiments are based on an iterative improvement flow and a solution can be oscillated in some cases. In these cases, we require several number of iterations to have a converged solution.

Table 3 Experimental results.

App.	FUs	$S_{\max}$	Algorithm	All dynamic energy [pJ]	Dynamic energy in registers [pJ]	Leak energy [pJ]	CT energy [pJ]	All energy [pJ]	Iterations	CPU time (ratio) [sec]
hal	Add × 1	5	Ref. [1]	119.049	108.218	28.3314	2.31617	147.380	2	382.32 (1.000)
	Sub × 1		Ref. [1] + CGCG	102.458	92.1862	28.3354	1.75765	130.793	2	394.70 (1.032)
	Mul × 2		Ref. [1] + FGCG	101.264	88.7507	28.3354	3.99872	129.599	2	394.70 (1.032)
	Com × 1		Ours w/o Section 3.1	86.4418	77.1209	29.6253	1.56093	116.067	2	419.99 (1.099)
			Ours	86.3667	77.1209	29.6253	1.48581	115.992	2	381.64 (0.998)
parker	Add × 2	10	Ref. [1]	28.3833	23.7773	68.0913	1.75087	96.4746	2	259.48 (1.000)
	Sub × 2		Ref. [1] + CGCG	26.2352	21.8553	68.0945	1.52478	94.3297	2	260.21 (1.003)
	Com × 1		Ref. [1] + FGCG	26.7604	18.0112	68.0945	5.89398	94.8549	2	260.21 (1.003)
			Ours w/o Section 3.1	21.7289	17.3088	67.9547	1.32697	89.6836	2	197.68 (0.762)
			Ours	21.7289	17.3088	67.9547	1.32697	89.6836	2	197.45 (0.761)
dct	Add × 4	10	Ref. [1]	199.459	132.922	24.4439	8.76961	223.903	3	868.33 (1.000)
	Mul × 4		Ref. [1] + CGCG	187.691	122.028	24.4554	7.89428	212.146	3	913.90 (1.052)
			Ref. [1] + FGCG	203.816	106.734	24.4554	39.3136	228.271	3	913.90 (1.052)
			Ours w/o Section 3.1	184.775	121.619	24.7141	8.18148	209.489	3	1,133.40 (1.305)
			Ours	174.917	113.327	26.2647	8.19640	201.182	3	1,167.48 (1.345)
dct	Add × 3	15	Ref. [1]	217.622	153.727	21.1666	10.0075	238.789	4	997.75 (1.000)
	Mul × 3		Ref. [1]+CGCG	207.997	146.126	21.1816	7.98265	229.179	4	999.68 (1.002)
			Ref. [1]+FGCG	219.559	121.599	21.1816	44.0723	240.741	4	999.68 (1.002)
			Ours w/o Section 3.1	200.278	136.085	22.0748	10.3092	222.353	10	2,300.39 (2.306)
			Ours	199.222	136.085	22.0748	9.25228	221.297	2	488.06 (0.489)
jacobi	Add × 2	15	Ref. [1]	202.975	112.942	77.3490	12.5441	280.324	2	293.82 (1.000)
	Sub × 1		Ref. [1] + CGCG	195.489	105.669	77.3591	12.3310	272.848	2	295.23 (1.005)
	Mul × 2		Ref. [1] + FGCG	209.365	91.7126	77.3591	40.1635	286.724	2	295.23 (1.005)
	Div × 2		Ours w/o Section 3.1	169.212	89.9171	79.1092	11.1721	248.321	2	308.09 (1.049)
			Ours	169.174	89.9171	79.0343	11.1586	248.208	2	273.61 (0.931)
fir	Add × 3	35	Ref. [1]	371.618	276.493	55.4682	19.6739	427.086	2	510.75 (1.000)
	Mul × 3		Ref. [1]+CGCG	344.971	249.569	55.5030	19.9510	400.474	2	554.51 (1.086)
			Ref. [1]+FGCG	364.275	214.678	55.5030	74.1458	419.778	2	554.51 (1.086)
			Ours w/o Section 3.1	332.851	239.651	55.8564	17.8229	388.707	2	531.03 (1.040)
			Ours	315.835	225.548	52.8838	13.7516	368.719	2	437.90 (0.857)
ewf3	Add × 4	45	Ref. [1]	590.185	396.598	111.197	34.3950	701.382	10	2,213.18 (1.000)
	Mul × 2		Ref. [1]+CGCG	549.058	358.927	111.257	30.9392	660.315	10	2,931.81 (1.325)
			Ref. [1]+FGCG	593.152	314.492	111.257	119.469	704.409	10	2,931.81 (1.325)
			Ours w/o Section 3.1	532.793	341.031	112.891	31.0000	645.684	10	2,519.56 (1.138)
			Ours	483.411	303.290	71.9863	25.6333	555.397	10	1,968.53 (0.889)
copy	Add × 3	165	Ref. [1]	69,469.4	67,816.4	3,107.74	1,106.34	72,577.1	10	4,980.24 (1.000)
	Sub × 1		Ref. [1] + CGCG	63,840.1	62,206.7	3,108.09	1,086.75	66,948.2	10	5,011.44 (1.006)
	Mul × 5		Ref. [1] + FGCG	77,119.8	51,606.0	3,108.09	2,4967.1	80,227.9	10	5,011.44 (1.006)
	Com × 1		Ours w/o Section 3.1	57,544.1	56,613.7	1,940.83	1,940.83	59,484.9	10	4,693.30 (0.942)
	Shi × 2		Ours	53,316.1	52,389.6	1,989.39	1,989.39	55,305.5	10	5,829.51 (1.171)

4.3 Optimality Evaluation

In order to evaluate optimality of our proposed concurrency-oriented scheduling algorithm, we picked up a scheduling/FU binding result from the output of our proposed algorithm on each application program and evaluated their optimality.

An optimal solution in our concurrency-oriented scheduling is that, as many registers as possible are active at the same time between control steps, so that we can apply a clock gating to non-active registers. We consider $Step(R)$ given by Eq. (8) in each huddle. Let $Max\_Step(R)$ be a maximum $Step(R)$ value when we consider a various subset R of the registers in the huddle. The larger the sum of $Max\_Step(R)$ over all the huddles is, the more registers are active at the same time. It leads to "better concurrency-oriented scheduling/FU binding." We can consider that our optimal solution is the one when the sum of $Max\_Step(R)$ over all the huddles is the largest.

Table 4 shows $Max\_Step(R)$ comparisons between our concurrency-oriented scheduling results and optimal results. Optimal results here were obtained by an exhaustive search. "Optimal ratio" is the ratio of the sum of $Max\_Step(R)$ in our proposed algorithm to the sum of $Max\_Step(R)$ in the optimal solution. It achieves an average of 92.2%. Roughly saying, the solutions of our proposed algorithm are close to optimal ones in small-sized and middle-sized application programs and we can say that our proposed algorithm is good enough to have concurrency-oriented scheduling/FU binding there. As one of our future challenges, we will develop a globally better concurrency-oriented scheduling algorithm including all the Strategies 1 to 4 discussed in Section 3.1 to have optimal solutions, even for large-sized application programs.

5. Conclusion

In this paper, we have proposed a high-level synthesis algorithm based on HDR architecture utilizing the huddle-based coarse-grain clock gatings considering concurrency-oriented scheduling/FU binding. Our proposed algorithm reduces all energy consumption by a maximum of 23.8% and an average of 14.4% compared with Ref. [1].

In the future, we not only apply clock gatings to HDR architectures but consider dynamic frequency control for them and propose its high-level synthesis algorithm. Moreover, setting at most one clock gating circuit per huddle is the first step in considering the clock gating in HDR architecture. In the future, we will develop a high-level synthesis algorithm that optimizes the numbers

Table 4 $Max\_Step(R)$ comparison between ours and optimal results.

App.	$S_{\max}$	Algorithm	$Max\_Step(R)$											Sum of	Optimal
			Hud.1	Hud.2	Hud.3	Hud.4	Hud.5	Hud.6	Hud.7	Hud.8	Hud.9	Hud.10	Hud.11	$Max\_Step(R)$	ratio
hal	5	Ours	3	4	8	4								19	1.000
		Optimal result	3	4	8	4								19	
parker	10	Ours	6											6	1.000
		Optimal result	6											6	
dct (+4*4)	10	Ours	15	18	9	3	3	14						62	0.912
		Optimal result	18	18	12	3	3	14						68	
dct (+3*3)	15	Ours	12	24	20	13	12							81	1.000
		Optimal result	12	24	20	13	12							81	
jacobi	20	Ours	32	10										42	0.913
		Optimal result	32	14										46	
fir	35	Ours	84	44	54	27								209	0.897
		Optimal result	96	50	54	33								233	
ewf3	45	Ours	40	46	50	23								159	0.909
		Optimal result	40	60	52	23								175	
copy	165	Ours	1,156	945	134	1,122	256	984	580	536	584	544	432	7,273	0.743
		Optimal result	1,972	1,755	134	1,972	256	1,008	580	536	584	552	438	9,787	

and the types of clock gatings in huddles.

Acknowledgments This work was supported partially by "Grant for Advanced Industrial Technology Development" from the New Energy and Industrial Technology Development Organization (NEDO) of Japan.

References

[1] Abe, S., Yanagisawa, M. and Togawa, N.: Energy-efficient high-level synthesis for HDR architectures, *IPSJ Trans. System LSI Design Methodologies*, Vol.5 (August Issue), pp.106–117 (2012).

[2] Akasaka, H., Yanagisawa, M. and Togawa, N.: Energy-efficient high-level synthesis for HDR architectures with clock gating, *Proc. IEEE International SoC Design Conference 2012*, pp.135–138 (2012).

[3] Cong, J., Fan, Y., Han, G., Yang, X. and Zhang, Z.: Architecture and synthesis for onchip multicycle communication, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol.23, No.4, pp.550–564 (2004).

[4] Emnett, F. and Biegel, M.: Power reduction through rtl clock gating, *Proc. Synopsys User Group Conference*, Citeseer (2000).

[5] Ohchi, A., Togawa, N., Yanagisawa, M. and Ohtsuki, T.: Floorplan-aware high-level synthesis for generalized distributed-register architectures, *IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences*, Vol.92, No.12, pp.3169–3179 (2009).

[6] Ohchi, A., Togawa, N., Yanagisawa, M. and Ohtsuki, T.: Performance-driven high-level synthesis with floorplan for GDR architectures and its evaluation, *Proc. IEEE International Symposium on Circuits and Systems*, pp.921–924 (2010).

[7] Ozaki, N., Amano, H., Nakamura, H., Usami, K., Namiki, M. and Kondo, M.: SLD-1 (Silent Large Datapath): A ultra low power reconfigurable accelerator, *Proc. IEEE Cool Chips XIV*, pp.9–17 (2011).

[8] Qurechi, S. and Sanjeev, K.: Power and performance optimization using multi-voltage, multi-threshold and clock gating for lowend microprocessors, *Proc. 1995 International Symposium on Low Power Design*, pp.9–14 (1995).

[9] Shin, J., Dawei, H., Petrick, B., Changku, H. and Leon, A.: A 40nm 16-core 128-thread SPARC SoC processor, *Proc. IEEE 2011 Solid State Circuits Conference*, pp.1–4 (2010).

[10] Singh, H. and Gajski, D.D.: A Design Methodology for Behavioral Level Power Exploration, Master's thesis, University of California, Irvine (1997).

[11] Yang, H. and Dung, L.: On multiple-voltage high-level synthesis using algorithmic transformations, *Proc. IEEE ASP-DAC*, pp.872–876 (Jan. 2005).

[12] Vittal, A. and Marek-Sadowska, M.: Low-power buffered clock tree design, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol.16, No.9, pp.965–974 (1997).

Hiroyuki Akasaka was born in 1989. He received his B.E. degree from Waseda University in 2012 in Computer Science. He is presently working toward a M.E. degree there. His research interest is high-level synthesis of LSIs.

Shin-ya Abe was born in 1988. He received his B.E. and M.E. degrees from Waseda University in 2011 and 2012, respectively, all in Computer Science. He is presently working toward a Dr.E. degree there. His research interests are design and verification of VLSI, especially high-level synthesis and energy efficiency design. He is a student member of IEICE.

Masao Yanagisawa was born in 1959. He received his B.E., M.E., and Dr.E. degrees from Waseda University in 1981, 1983, and 1986, respectively, all in Electrical Engineering. He was with University of California, Berkeley from 1986 through 1987. In 1987, he joined Takushoku University. In 1991, he left Takushoku University and joined Waseda University, where he is presently a Professor in the Department of Electronic and Photonic Systems. His research interests are combinatorics and graph theory, computational geometry, VLSI design and verification, and network analysis and design. He is a member of IEEE and ACM and a fellow of IEICE.

Nozomu Togawa was born in 1970. He received his B.E., M.E., and Dr.E. degrees from Waseda University in 1992, 1994, and 1997, respectively, all in Electrical Engineering. He is presently a Professor in the Department of Computer Science and Engineering, Waseda University. His research interests are LSI design, graph theory, and computational geometry. He is a member of IEEE and IEICE.

(Recommended by Associate Editor: *Takashi Takenaka*)