Energy-efficient High-level Synthesis Algorithms for Floorplan-driven SoC Architectures

フロアプラン指向集積回路アーキテクチャを対象とした低エネルギー高位合成に関する研究

February 2015

Shinya ABE

阿部 晋矢

Energy-efficient High-level Synthesis Algorithms for Floorplan-driven SoC Architectures

フロアプラン指向集積回路アーキテクチャを対象とした低エネルギー高位合成に関する研究

February 2015

Waseda University

Graduate School of Fundamental Science and Engineering

Department of Computer Science and Engineering

Research on Information System Design

Shinya ABE

阿部 晋矢

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recently, system-on-a-chip (SoC), which contains all the necessary electronic circuits and parts for a system on a single integrated circuit, is widely used in various ICT devices and becomes the important ICT core technologies. SoC design has already been switching from large-lot production of narrow-ranging products to small-lot production of wider-ranging products. Therefore, SoC designers should reduce non-recurring engineering (NRE) costs rather than recurring manufacturing costs. Increasing design abstraction level is one of the most effective strategies for improvements in design productivity. Fig. 1.1 shows estimated design cost of the SoC low-power (SOC-LP) PDA [12]. In Fig. 1.1, the orange line shows the design cost of register transfer level (RTL) methodology, which is currently used, and the green line shows that of future methodology with higher level of abstraction. In RTL design, circuits are described by hardware description language (HDL), such as Verilog-HDL and VHDL, where designers consider something specific to hardware such as registers and clock synchronization. High-level synthesis (HLS) is the LSI design automation technique which obtains circuits from a behavior level. HLS translates behavior level description which is described by higher abstract language, such as C and C++, into RTL design. NRE costs can be reduced by HLS tools since hardware specific description in RTL design automatically generated by the algorithms in the HLS tools. HLS tools and the algorithms are evaluated by the performance of the output SoC. In recent SoC design, energy-efficiency has become one of the most important factors due to the growth of battery-powered portable devices. For obtaining low-energy SoC design, (1) HLS should deal with energy-efficient LSI design techniques and (2) module floorplanning in SoC should be considered during HLS.

There are several energy-efficient LSI design techniques such as multiple supply voltages (MSV) [41], dynamic multiple supply voltages (DMSV) [24,33], and multiple clock domains (MCD) [5,30]. Since any design decision made at earlier stages

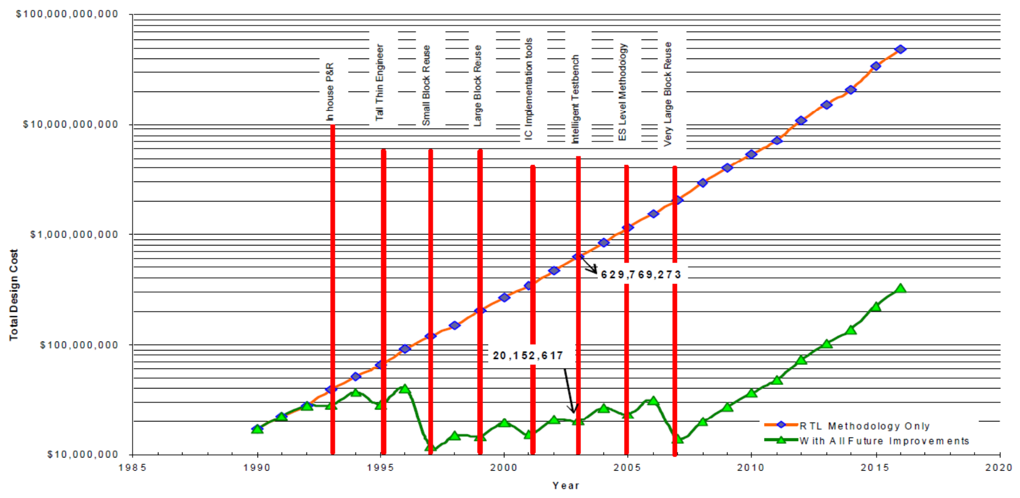


Figure 1.1: Impact of Design Technology on SOC LP-PDA Implementation Cost [12].

has higher impacts on the final result, these energy-efficient techniques should be applied during HLS. Several energy-aware high-level synthesis algorithms have been proposed which deal with MSV [4, 18, 25, 26, 34, 35, 38, 47, 48], DMSV [6], and MCD [23]. However, the conventional methods cannot reduce energy effectively because they do not consider floorplanning. First, they cannot estimate circuit delay accurately. As device feature size decreases, an interconnection delay, which is a delay necessary for the communication between modules inside an LSI chip, becomes the dominant factor of circuit total delay. Since the existing works only consider a gate delay which is a delay necessary for the transistors to charge or discharge, energy consumption of circuits cannot reduce as expected or the circuits may be inoperable. Second, they cannot estimate energy consumption sufficiently. For example, the energy consumption of clock signal cannot be estimated when the module placement inside an LSI chip is unknown. In order to solve the problems and reduce further energy consumption, module floorplanning should be integrated into HLS algorithms.

Interconnection delay is another important issue in HLS because interconnection delay accounts for a large percentage of circuit delay as device feature size decreases. There are several HLS algorithms which consider interconnection delay effects [8, 17, 19, 31, 32]. They are not based on a traditional centralized-register architecture, but they are based on distributed-register architecture (DR architecture) families. In DR architectures, chip area is divided into sufficiently small partitions such that the interconnection delay inside each partition can be assumed to be zero. The interconnection delay between the partitions is estimated by placement information which can be obtained by floorplanning the partitions during HLS. Furthermore, more various kinds of energy consumption, such as clock signal energy and interconnection energy, can be estimated if DR architectures are used. However, the objective of conventional DR architectures and synthesis algorithms is the optimization of circuit latency and they do not consider energy efficiency. Conventional methods are not suitable to adopt energy-efficient LSI design techniques such as MSV, DMSV, and MCD.

In this dissertation, I propose new floorplan-driven SoC architectures to which energy-efficient LSI design techniques, such as MSV, DMSV, and MCD, are easily applicable. Furthermore, the associated HLS algorithms are proposed for energy reduction based on the proposed the architectures. The proposed algorithms can reflect floorplanning information in HLS by using iterative synthesis flows. By using a floorplanning result, interconnection dela
estimated, and then optimized supply voltages and/or clock periods can be assigned for energy reduction. Experimental results show that the proposed methods achieve 22.4% energy-saving by applying MSV, 43.9% energy-saving by applying DMSV, and 57.0% energy-saving by applying MCD and MSV compared with the existing methods.

This dissertation is organized as follows:

Chapter 2 [Related Works] lated works. First, I preview the low-power and low-energy LSI design techniques and energy-efficient HLS algorithms which consider the techniques such as MSV, power gating (PG), DMSV, and MCD. Next, I preview the DR architecture families and the HLS algorithms for the architectures which can consider floorplanning during HLS. **Chapter 3 [A Multiple Supply Voltages Aware High-level Synthesis Algorithm for HDR Architecture]** proposes huddle-based distributed-register architecture (HDR architecture) and an HLS algorithm associated with HDR architecture. HDR architecture divides chip area into several partitions called huddles. Huddles enable us to estimate interconnection dela ly and assign supply voltages effectively. I propose the new HLS algorithm which can automatically apply MSV to SoC by assigning voltages to huddles and outputs energy-efficient SoC designs. Experimental results show that the proposed method achieves 22.4% energy-saving compared with the conventional methods. **Chapter 4 [MH^4: Multiple Supply Voltages Aware High-speed and High-efficiency High-level Synthesis Algorithm for HDR architecture]** proposes an improved HLS algorithm for HDR architecture called MH^4. The algorithm proposed in Chapter 3 has the two severe problems: (A) the huddle-area and interconnection-delay oscillation during iterations and (B) the insufficient huddle construction methods. I propose three new techniques, virtual area estimation, virtual area adaptation, and floorplanning-directed huddling, to resolve the problems (A) and (B) and then proposes a new multiple-supply-voltages aware high-speed and high-efficiency high-level synthesis algorithm for HDR architecture. Experimental results show that the proposed algorithm achieves 29.1% run-time-saving compared with the algorithm in Chapter 3, and successfully obtains a solution which cannot converge when the algorithm in Chapter 3 is used. **Chapter 5** [**SAAV: Dynamic Multiple Supply Voltages Aware High-level Synthesis Algorithm for AVHDR Architecture**] proposes adaptive voltage huddle-based distributed-register architecture (AVHDR architecture) and an HLS algorithm associated with AVHDR architecture. First, I propose a new distributed-register architecture called AVHDR architecture which can apply DMSV. Next, I propose a new HLS algorithm for AVHDR architecture called SAAV. In AVHDR architecture and SAAV, low supply voltages can be assigned to non-critical operations and leakage power can be cut off through PG. Experimental results show that the proposed method achieves 43.9% energy-saving compared with the conventional methods. **Chapter 6** [**SAMCID: Multiple Clock Domains Aware High-level Synthesis Algorithm for HDR-mcd Architecture**] proposes HDR-mcd architecture and an HLS algorithm associated with HDR-mcd architecture. First, I propose a new distributed-register architecture called HDR-mcd architecture which can apply MCD. Next, I propose a new HLS algorithm for HDR-mcd architecture called SAMCID. Experimental results show that the proposed method which only considers MCD achieves 32.5% energy-saving compared with the conventional methods. Furthermore, the proposed method which can apply MCD and MSV simultaneously achieves 57.0% energy-saving compared with the conventional methods. **Chapter 7** [**Conclusion**] summarizes the research and indicates future works.

Chapter 2

Related Works

2.1 Introduction

In this section, related works have been discussed. First, I preview the low-power and low-energy LSI design techniques and energy-efficient HLS algorithms which consider the techniques such as MSV, power gating (PG), DMSV, and MCD. Next, I preview the DR architecture families and the HLS algorithms for the architectures which can consider floorplanning during HLS.

2.2 Energy-efficient LSI Design Techniques

Power consumption in CMOS circuits [45] is:

$$P_{total} = P_{dynamic} + P_{static} \tag{2.1}$$

where $P_{dynamic}$ is dynamic power consumption and P_{static} is static power consumption. $P_{dynamic}$ is:

$$P_{dynamic} = P_{switching} + P_{shortcircuit} \tag{2.2}$$

where $P_{switching}$ is power consumption of charging and discharging load capacitances as gate switch and $P_{shortcircuit}$ is that of short-circuit current while both pMOS and nMOS stacks are partially ON. P_{static} is:

$$P_{static} = (I_{sub} + I_{gate} + I_{junct} + I_{contention})V_{DD} \tag{2.3}$$

where I_{sub} is subthreshold leakage current through OFF transistors, I_{gate} is gate leakage current through gate dielectric, I_{junct} is junction leakage current from source/drain diffusions, and $I_{contention}$ contention current in rationed circuits. When the chip is doing useful work, P_{total} is usually dominated by $P_{switching}$. $P_{switching}$ is expressed as:

$$P_{switching} = \alpha C V_{DD}^2 f \tag{2.4}$$

where α is a switching activity factor, C is the capacitance of a transistor, V_{DD} is a voltage source, and f is a clock frequency. LSI design techniques for energy and/or power reduction aim to reduce the value of each variable in Eqs. (2.1), (2.2), (2.3), and (2.4).

Multiple supply voltages (MSV) techniques and high-level synthesis considering MSV

Multiple supply voltages (MSV) technique [41] is an energy-efficient LSI design which focuses on V_{DD} in Eq. (2.3) and Eq. (2.4). Supply voltage reduction is one of the most effective techniques, but the delay increases with reducing V_{DD}. In order to reduce energy consumption without degrading the performance, we divide a chip into multiple voltage domains. In MSV, high supply voltages are assigned to elements on critical paths and low supply voltages are assigned to elements on the non-critical paths. Since $P_{switching}$ is proportional to the square of V_{DD}, it is the most effective for dynamic and total energy reduction that V_{DD} is reduced. Furthermore, P_{static} can be reduced by the V_{DD} reduction because P_{static} is directly proportional to V_{DD}. We have to insert level converters (level shifters) when data are exchanged between different voltage domains [2, 40, 41]. Since inserted level converters have an overhead of area, delay, and energy consumption, we should consider the overheads when MSV technique is utilized. Fig. 2.1 shows a level converter circuit [41].

Several energy-aware high-level synthesis algorithms have been proposed which deal with multiple supply voltages [4, 18, 21, 25–27, 34, 35, 37, 38, 46–48]. They reduce energy consumption under time constraints [34], resource constraint [35], and both of them [18, 25, 26, 38, 47, 48]. In terms of algorithms, some of them [4, 18, 25] do the exact optimization using integer linear programming (ILP) and others [25, 26, 38, 47, 48] are based on heuristic algorithms. However, only a few of them consider the level converter overheads. [47, 48] assigned supply voltages for acquiring the energy overheads of level converters. All of the conventional methods always add or ignore the delay of level converters. As far as I know, there are no methods which accurately consider the delay overheads of level converters. Furthermore all of the conventional high-level synthesis algorithms do not consider interconnection dela . Therefore, energy consumption of circuits cannot be reduced as expected or the circuits may be inoperable based on the existing works.

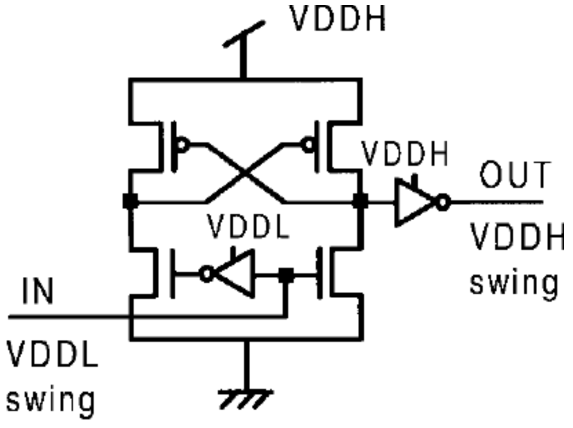


Figure 2.1: A level converter circuit ([41]).

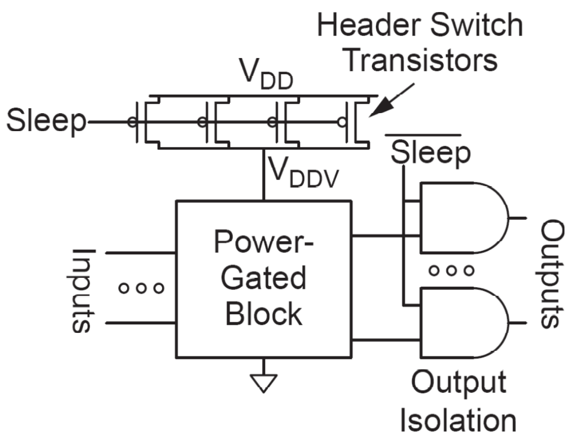


Figure 2.2: Power gating [45].

Power gating (PG) and dynamic multiple supply voltages (DMSV) techniques, and high-level synthesis considering PG and DMSV

Power gating (PG) technique focuses on P_{static}. Switch transistors cut off the electronic connection between a power gating block and the power rail and/or the ground. Fig. 2.2 shows an example of the PG circuit [45]. Since "V_{DDV}" equals virtually zero when "Sleep" is ON, P_{static} of "Power-Gated Block" can be reduced. Several energy-efficient high-level synthesis methods which deal with PG have been proposed [7, 10]. However, they do not consider interconnection dela
dynamic multiple supply voltages described later.

The dynamic multiple supply voltages (DMSV) technique has been proposed

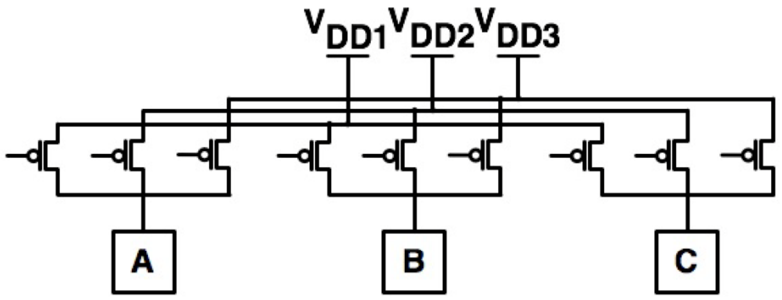


Figure 2.3: PDVS architecture [33].

in recent years [24, 33] in which non-critical components are assigned to lower supply voltages to reduce dynamic energy consumption. In addition, multiple supply voltages can be combined with power gating for leakage energy reduction. Supply voltages are changed dynamically at run-time using switch transistors. The panoptic dynamic voltage scaling (PDVS) is an ASIC architecture utilizing with DMSV and Fig. 2.3 shows the PDVS architecture [33]. An HLS algorithm which deals with dynamic multiple supply voltages has been proposed in [6]. However, [6] only consider operation binding and voltage assignment. For further energy reduction, scheduling and floorplanning problem should be optimized.

Multiple clock domains (MCD) and high-level synthesis considering MCD

Multiple clock domains (MCD) technique [5,30] focuses on switching activity α of the clock and related registers in Eq. (2.4). [11] found that roughly one-third of microprocessor power is spent on the clock. Furthermore, as Table DESN5 in ITRS 2011 [16] describes *"Departure from fully synchronous design paradigm needed for power reduction, latency insensitivity, variation-tolerance,"* multiple clock domains (MCD) techniques have been growing in importance in recent energy-efficient LSI design.

There are two types of MCD techniques: (1) One can utilize arbitrary clock frequencies but synchronization circuits are needed for the communication between different clock domains. (2) The other can communicate with different clock domains without synchronization circuits but can only utilize regular clock frequencies which follow certain fixed patterns. Global asynchronous local synchronous (GALS) [5] is well known as an example of Type (1). GALS can reduce energy consumption effectively when optimal clocks are assigned to different cores in multiprocessor SoC or network-on-chip. Although GALS is often applied to multi-

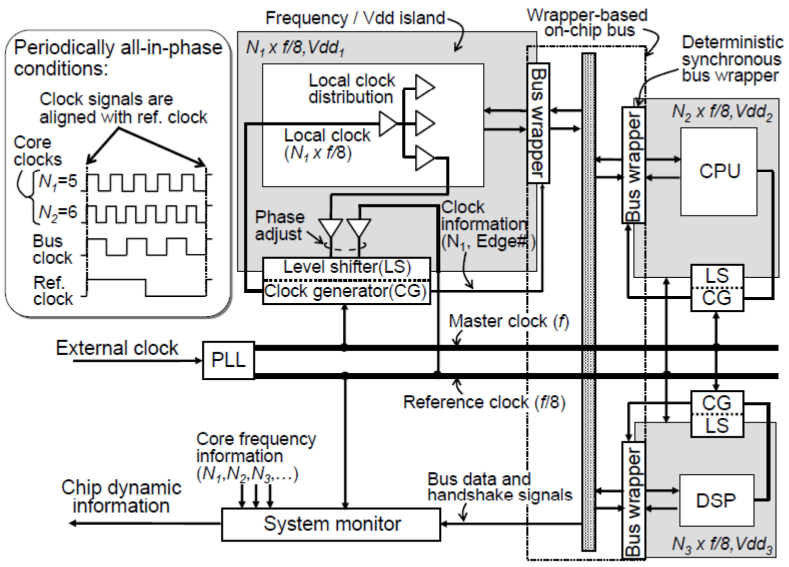


Figure 2.4: The architecture of the periodically all-in-phase [30].

processor SoC or network-on-chip, there are few high-level synthesis algorithms considering GALS. In [23], an HLS algorithm by dealing with GALS has been proposed. However, [23] can consider only two clock frequencies and cannot deal with the clock domain division process.

Periodically all-in-phase proposed in [30] is Type (2) based MCD technique for a smaller-scale circuit. In periodically all-in-phase technique, the clock periods of the local clock signals are the integer multiples of reference clock signals and the local clock signals can synchronize at the periods of common multiples without any synchronization circuits. Fig. 2.4 shows the architecture of the periodically all-in-phase technique. As far as I know there are no existing HLS algorithms considering periodically all-in-phase.

There are the technique which can dynamically change clock frequency at run-time, such as DVFS [3]. However, there are no conventional high-level synthesis algorithms which consider DVFS. Scheduling algorithms and binding algorithms of high-level synthesis is based on a clock periods which are fixed at run-time. Therefore the problem definition which deal with frequency scaling is too difficult. In this dissertation, the dynamically change of clock frequency is not considered as well as conventional high-level synthesis algorithms.

2.3 Distributed-register Architectures

Interconnection delay is another important issue in HLS because interconnection delay accounts for a large percentage of circuit delay as device feature size decreases. Traditional HLS algorithms assume that the target circuit has a centralized register file and a global controllers. The architecture configuration aimed at the minimization of the number of registers and/or the total circuit area. However, the interconnection delay between a functional unit and the register file may become the biggest fraction of the total circuit delay. A circuit which is got by a traditional HLS algorithm may not work as expected or the latency and the associated energy consumption may increase. In order to deal with the interconnection delay increase, we need new circuit architectures unlike the traditional centralized-register architecture. There are several HLS algorithms which consider interconnection dela [8, 9, 17, 19, 31, 32]. They are not based on a traditional centralized-register architecture, but they are based on distributed-register architecture (DR architecture) families. In DR architectures, chip area is divided into sufficiently small partitions such that the interconnection dela
partition can be assumed to be zero. On the other hand, the inter-partition data transfer is realized through multi-cycle interconnect communication between local registers. The interconnection delay between partitions is estimated by placement
lanning during HLS.

A basic DR architecture has been proposed in [17, 19]. Fig. 2.5 shows the architecture ([17]). In DR architecture, each functional unit performs computation by using its dedicated local registers. Since the wire length between the functional unit and its local registers is small enough the interconnection delay between the functional unit and its local registers can be ignored. They proposed an HLS algorithm for the DR architecture in [19] at first. Next, they proposed an iteration based HLS algorithm shown in Fig. 2.5(b) ([17]). Because placement results and scheduling/binding results influence each other, iterative refinement flow in [17] can obtain higher performance on resulting circuits. However, DR architecture has the large overhead of area and energy because this architecture requires local registers placed for each functional unit.

A regular distributed-register architecture (RDR architecture) is proposed in [8]. Fig. 2.6 shows an example of RDR architecture and HLS algorithm for RDR architecture called MCAS proposed in [8]. RDR architecture divides a chip into uniform-sized islands and arranges functional units, a register file, and a controller in each island. Inside an island, interconnection delay can be ignored by using local registers. By introducing uniform-sized islands, RDR architecture realizes multi-cycle interconnect communication during inter-island data transfer. In RDR

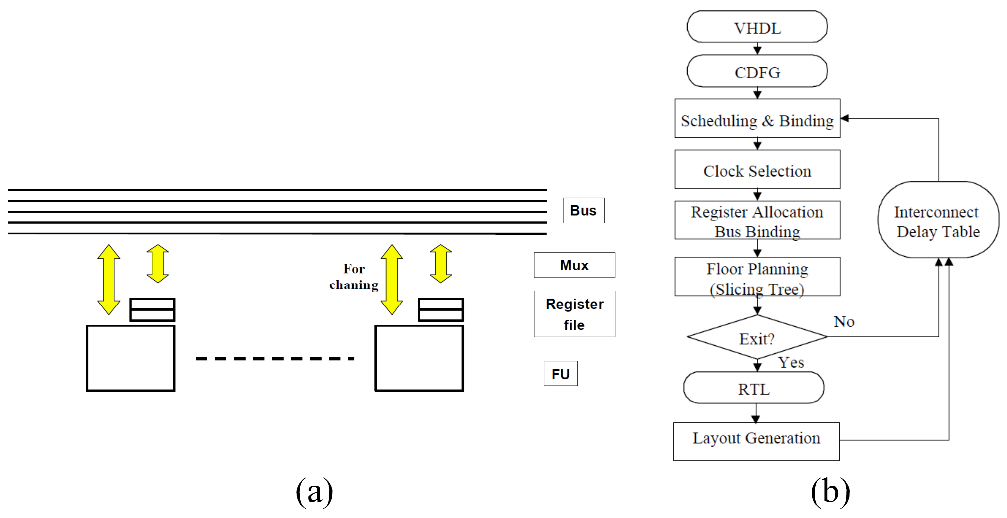


Figure 2.5: Distributed-register architecture and a synthesis algorithm proposed in [17]. (a) The architecture. (b) The algorithm.

architecture, it is very easy to predict interconnection delays even in high-level synthesis stage since an entire chip is divided into uniform-sized islands. In addition, it is easy to cope with the module addition/deletion in an island since the island abstracts modules inside. RDR architecture also has several improved architectures: RDR-Pipe [8] decreases the number of connections by adding a new module; DRFM [9] is proposed to implement the RDR architecture on FPGA devices; a fault-secure high-level synthesis algorithm on RDR architectures proposed in [39] improve soft error resistance by adding new modules to vacant islands. However, RDR architecture has the area and energy overheads since it divides an entire chip into uniform-sized islands. Area overhead causes useless modules and will increase interconnection delays. Thus, [8] itself describes *"The RDR architecture is not suitable (or necessary) for low-frequency (and lowpower) designs as due to its unnecessary area (and power) overhead."*

A generalized distributed-register architecture (GDR architecture) is proposed in [32]. GDR architecture realizes multi-cycle interconnect communications by preparing two kinds of registers: local registers and shared register groups, and two kinds of controllers: global controllers and local controllers (Fig. 2.7(a)). In [32], scheduling/binding as well as floorplanning are simultaneously optimized by using iterative synthesis flow (Fig. 2.7(b)). Since functional units, shared/local registers, and global/local controllers can be very flexibly arranged on GDR architecture, it can obtain high performance design. However, this flexibly can become a disadvantage that its associated high-level synthesis problems are too complex [32] where

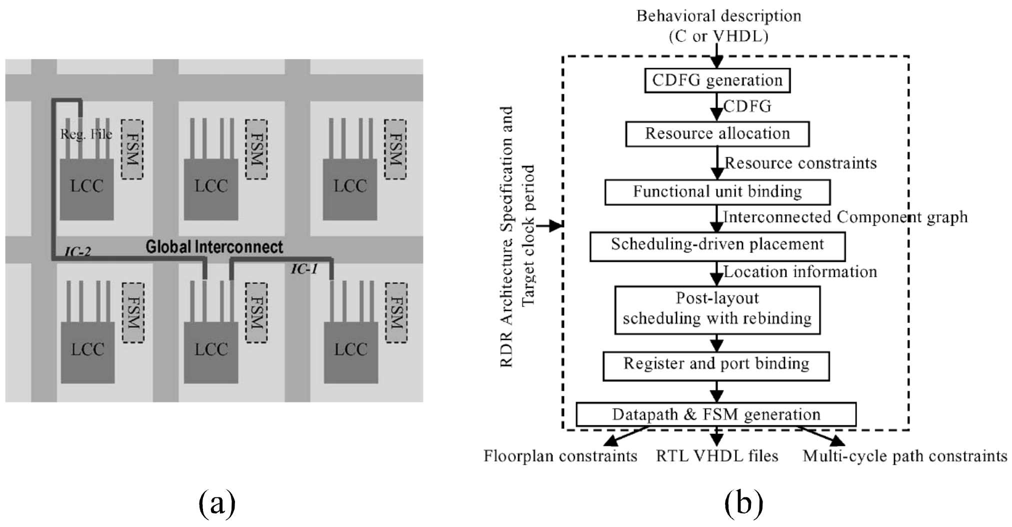


Figure 2.6: RDR architecture and a synthesis algorithm called MCAS proposed in [8]. (a) 2×3 island-based RDR architecture. (b) MCAS for RDR architecture.

we have to consider which register is shared or local as well as which controller is global or local. If we apply energy-efficient LSI design techniques including multiple supply voltages to GDR architecture, it is necessary to add new modules such as level converters. It definitely increases the complexity of the high-level synthesis problem and it must be very difficult to cope with energy-saving techniques in GDR architecture synthesis.

According to the above discussion, conventional DR architectures and synthesis algorithms are not suitable to adopt energy-efficient LSI design techniques for energy reduction. However, DR architectures can reduce total circuit delay more than the traditional centralized-register architecture. Since energy-saving techniques reduce energy consumption at the expense of circuit delay, DR architectures can reduce energy consumption more than the centralized-register architecture by applying energy-saving techniques. DR architectures suitable for applying energy-saving techniques are the ones which have small area and low energy consumption and in which it is easy to add new modules. Based on the discussion above, new DR architectures and HLS algorithms are proposed for energy reduction later.

2.4 Conclusion

In this chapter, the related works for energy-reduction and interconnection delay optimization in high-level LSI design are briefly discussed. From the point of view

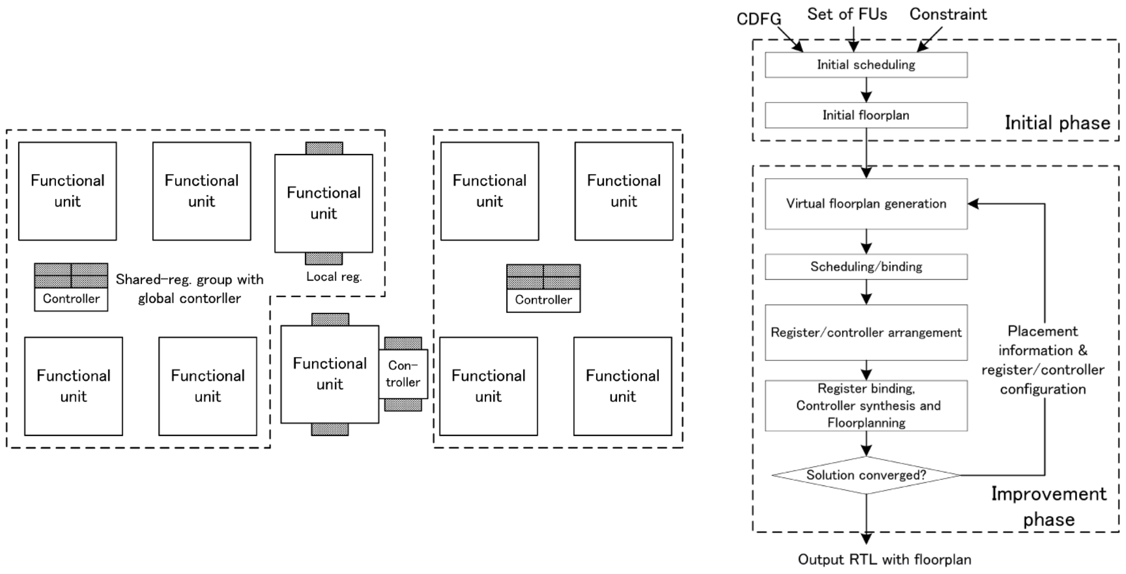

Functional unit
Shared-reg. group with global contorller
Controller
Local reg.
Con-troller
CDFG
Set of FUs
Constraint
Initial scheduling
Initial floorplan
Initial phase
Virtual floorplan generation
Scheduling/binding
Register/controller arrangement
Register binding, Controller synthesis and Floorplanning
Solution converged?
Placement information & register/controller configuration
Improvement phase
Output RTL with floorplan
(a)
(b)

Figure 2.7: GDR architecture and a synthesis algorithm for the architecture proposed in [32]. (a) a GDR architecture. (b) An HLS algorithm for GDR architecture.

of energy-efficient LSI design, conventional HLS algorithms have two problems:
(1) Most energy-efficient HLS algorithms did not consider floorplanning during high-level synthesis,
(2) Existing DR architectures are not suitable to adopt energy-efficient LSI design techniques for energy reduction. In the following sections, I propose new distributed-register architectures called huddle-based distributed-register architecture (HDR architecture) families to adopt energy-efficient LSI design techniques such as MSV, DMSV, and MCD. Furthermore, I propose new energy-efficient HLS algorithms for the architectures.

Chapter 3

A Multiple Supply Voltages Aware High-level Synthesis Algorithm for HDR Architecture

3.1 Introduction

In this Chapter, a high-level synthesis algorithm considering energy-efficiency and interconnect delays simultaneously is proposed. First, a *huddle-based distributed-register architecture (HDR architecture)*, which is one of the distributed-register architectures (DR architectures) focusing on energy-efficiency is proposed. In HDR architecture, functional units, registers, controllers, and level converters are abstracted into a *non*-uniform island, which is called a *huddle*. Second, a high-level synthesis algorithm which is associated with an HDR architecture is proposed. The proposed algorithm can reflect floorplan information in scheduling/binding by using iterative synthesis flow. At that time, modules which are placed close to each other are *huddled* into a non-uniform island. Then each huddle shares functional units, registers, controllers, and level converters. Interconnection delays inside a huddle can be ignored by using local registers inside and HDR also support multi-cycle interconnect communications during inter-huddle data transfer. Every huddle has its own supply voltage; low supply voltages are assigned to huddles on non-critical paths and high supply voltages are assigned to huddles on critical paths. Experimental results show that the proposed algorithm achieves 22.4% energy-saving compared with the conventional distributed-register architectures and conventional algorithms.

3.2 HDR Architecture

In this section, recent DR architectures, such as GDR architecture and RDR architecture, are briefly reviewed and it is pointed out that they are not suitable for applying energy-saving techniques. After that, a new distributed-register architecture called HDR architecture is proposed.

Generalized distributed-register architecture (GDR architecture) proposed in [32] can synthesize high-performance and small-area circuits by introducing shared / local registers and global / local controllers. Since functional units, shared / local registers, and global / local controllers can be very flexibly arranged on a GDR architecture, it can obtain high performance design. However, this flexibly can become a disadvantage that its associated high-level synthesis problems are too complex [32] where we have to consider which register is shared or local as well as which controller is global or local. In order to realize power reduction using multiple supply voltages, it is necessary to add new modules, such as level converters when changing voltages. It definitely increases the complexity of the high-level synthesis problem and it must be very difficult to cope with energy-saving techniques in GDR architecture synthesis.

Regular distributed-register architecture (RDR architecture) proposed in [8] can predict interconnection delays in high-level synthesis accurately by dividing a chip into uniform-sized islands. It arranges functional units, local registers, and a controller inside an island, and multi-cycle interconnect communication during inter-island data transfer can be also realized. In RDR architecture, a module can be easily added to an island since it abstracts each module inside. [39] realizes fault-secure high-level synthesis using RDR architecture based on adding functional units to an island. On the other hand, RDR architecture has significant area overhead since they divide a chip into uniform-sized islands. Area overhead may lead the increase of useless modules as well as interconnection delays. We can say that RDR architecture is not suitable for applying energy-saving techniques, either.

DR architectures suitable for applying energy-saving techniques are the ones which have small area and low power consumption and in which it is easy to add new modules. Based on the discussion above, a huddle-based distributed-register architecture (HDR architecture), which is one of the distributed-register architecture but has energy-efficiency combining the advantages of RDR architecture and GDR architecture, is proposed.

HDR architecture introduces a non-uniform sized island called a *huddle* into GDR architecture in which each module inside is abstracted. As seen in RDR architecture, it is very easy to add new modules into a non-uniform sized island. The huddle has non-uniform rectangular area determined by clock period constraints

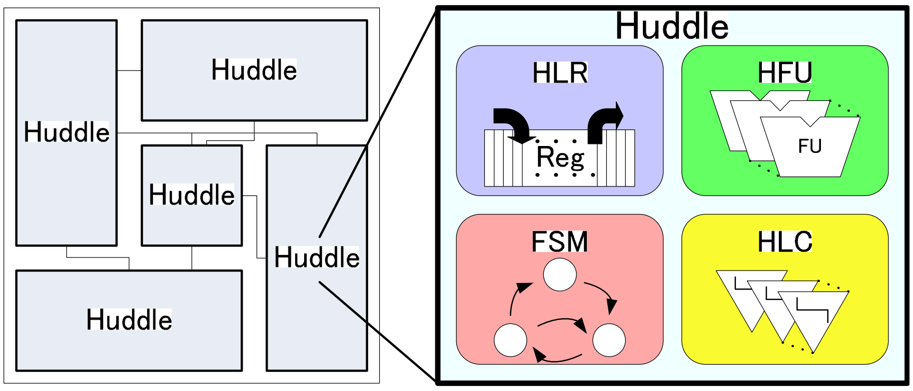


Figure 3.1: A huddle configuration.

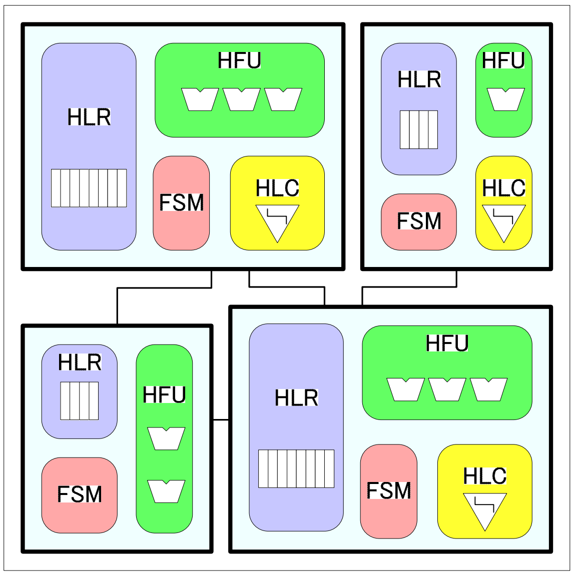


Figure 3.2: HDR architecture.

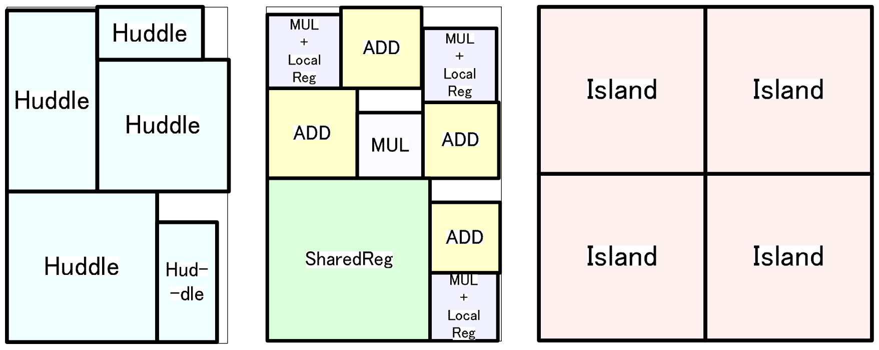
which includes functional units, registers, controllers, and level converters. Since the huddles have non-uniform rectangular area, HDR architecture can be synthesized with small area and small energy consumption.

Fig. 3.1 shows a huddle configuration. A huddle h consists of the following components:

Huddled Local Registers (HLRs): Dedicated local registers in h.

Huddled Functional Units (HFUs): Dedicated functional units in h. HFUs can only access the HLRs in h.

Finite State Machine (FSM): A dedicated controller in h. FSM controls the HFU and the HLR in h.



(a) HDR. (b) GDR. (c) RDR.

Figure 3.3: Experimental results of HDR, GDR [32], and RDR [8] when DCT is synthesized.

Huddled Level Converters (HLCs): Dedicated level converters in h. HLCs are used during inter-huddle data transfer across different voltage huddles.

Huddles with different size and different components can be placed as in Fig. 3.2. Interconnection delays can be ignored by using HLRs inside a huddle and multi-cycle interconnect communication during inter-huddle data transfer can be realized. HLCs are used during multi-cycle interconnect communication across different voltage huddles.

Example 3.1. *Fig. 3.3 shows the high-level synthesis results of HDR architecture, GDR architecture [32], and RDR architecture [8]. DCT is used as a benchmark application. HDR architecture drastically reduces synthesis complexity as compared to GDR architecture because HDR architecture abstracts modules by introducing huddles. HDR architecture also reduces area overhead as compared to RDR architecture because huddles have non-uniform rectangular area.* □

3.3 Problem Definition

A control-data flow graph (CDFG) $G(N, E)$ is a directed graph, where a node set N is composed of an operation node set N_o and a branching control node set N_c (start and end nodes of conditional branches), and an edge set E is composed of a data-flow edge set E_d and a control-flow edge E_c set. T_{clk} refers to a clock period constraint and S_{max} refers to a control step constraint.

Let $F = \{f_1, \cdots, f_p\}$ be a set of 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 in $S_f(f_i)$ steps.

Let $H = \{h_1, \cdots, h_q\}$ be a set of huddles in the HDR architecture. Each functional units are bound to any one of the huddles and the binding is defined by a function $Hud : F \to H$. $Hud(f_i)$ is the huddle to which f_i is bound. $F(h_j)$ is a set of functional units which are bound to h_j. $D_{reg}(h_j)$ is a delay of HLRs in h_j.

The three supply voltages, v_l, v_m, and v_h $(v_l < v_m < v_h)$, which are assigned to each huddle, are considered. $V(h_j)$ is a supply voltage which are assigned to the huddle h_j. $D_{lc}(v_l, v_m)$ is a delay of a level converter which changes the voltage from v_l to v_m. Likewise, $D_{lc}(v_l, v_h)$, $D_{lc}(v_m, v_l)$, and so on can be defined.

$Slack(f_j)$ is defined by:

$$Slack(f_j) = T_{clk} \cdot S_f(f_i) - D_f(f_i). \tag{3.1}$$

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

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

$$2 \cdot D_w(W(h_j) + H(h_j)) + D_{reg}(h_j) \le \min_{f_i \in F(h_j)} \{Slack(f_i)\} \tag{3.2}$$

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 whose length is x. In the proposed algorithm, we obtain the value of $(W(h_j) + H(h_j))$ so that it satisfies the huddle size constraint and determine $W(h_j)$ and $H(h_j)$ by using the aspect ratio predefined for each huddle.

Let $Dist(h_j, h_k)$ be the Manhattan distance between the center of huddles h_j and h_k. Then $D_w(Dist(h_j, h_k))$ shows the interconnection delay between them. 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 HLRs in h_k which is defined by:

$$Tr(f_i, h_k) = D_w(Dist(h_j, h_k)) + D_{lc}(V(h_j), V(h_k)) + D_{reg}(h_k). \tag{3.3}$$

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

$$DT(f_i, h_k) = \begin{cases} 0, & (Slack(f_i) \ge Tr(f_i, h_k)) \\ \lceil Tr(f_i, h_k)/T_{clk} \rceil . & (Slack(f_i) < Tr(f_i, h_k)) \end{cases} \tag{3.4}$$

In the case of $Slack(f_i) \ge Tr(f_i, h_k)$, the functional unit f_i directly stores its output in the register file of huddle h_k. Thus, the data transfer requires no extra

Table 3.1: Delay/energy of functional units.

	Adder	Multiplier
v_h	1 ns/144 fJ	2 ns/1440 fJ
v_m	2 ns/100 fJ	4 ns/1000 fJ
v_l	4 ns/64 fJ	8 ns/640 fJ

Table 3.2: Level converter delays.

	v_h	v_m	v_l
v_h	-	0.5 ns	1.0 ns
v_m	0.5 ns	-	2.0 ns
v_l	1.0 ns	2.0 ns	-

cycles. In the case of $Slack(f_i) < Tr(f_i, h_k)$, the functional unit f_i first stores its output into the local register file in its huddle h_j $(= Hud(f_i))$. In the next cycle, the data transfer from the huddle h_j to the huddle h_k starts. The data transfer requires $\lceil Tr(f_i, h_k)/T_{clk} \rceil$ cycles. Then the data transfer table DT can be defined by a $p \times q$ matrix whose (i, k)-element is expressed by $DT(f_i, h_k)$.

Example 3.2. *Fig. 3.4 and Fig. 3.5 show an example of HDR architecture with a clock period constraint $T_{clk} = 3\,\text{ns}$ and a control step constraint $S_{max} = 5$. Table 3.1 shows delay and energy consumption of each functional unit. Table 3.2 shows delay of each level converter.*

As in Fig. 3.4, the input functional units are $f_1 = \text{MUL1}$, $f_2 = \text{MUL2}$, $f_3 = \text{MUL3}$, and $f_4 = \text{ADD1}$. In this example, we have huddle configurations of $F(h_1) = \{f_1\}$, $F(h_2) = \{f_2\}$, and $F(h_3) = \{f_3, f_4\}$. Voltages are assigned to the huddles as in Fig. 3.4: $V(h_1) = V(h_2) = v_h$ and $V(h_3) = v_m$.

Let $D_w(Dist(h_1, h_3)) = 1\,\text{ns}$ and $D_{reg}(h_3) = 0.5\,\text{ns}$. $Slack(f_1)$ can be calculated as:

$$Slack(f_1) = 3\text{ns} \times 1 - 2\text{ns} = 1\text{ns}.$$

$Tr(f_1, h_3)$ is calculated by

$$Tr(f_1, h_3) = 1\text{ns} + 0.5\text{ns} + 0.5\text{ns} = 2\text{ns}.$$

Since $Slack(f_1) < Tr(f_1, h_3)$, $DT(f_1, h_3)$ can be calculated by:

$$DT(f_1, h_3) = \lceil 2/3 \rceil = 1.$$

Data transfer from the functional unit f_1 to the huddle h_3 requires 1 clock cycle. As in Fig. 3.5, the data transfer from the node 4 (*) to the node 7 (+) requires extra control step (CS3) for multi-cycle interconnect communication. □

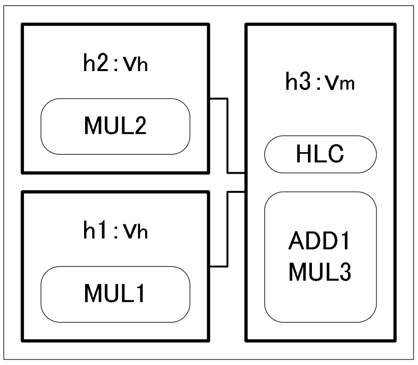


Figure 3.4: An example of HDR architecture configuration.

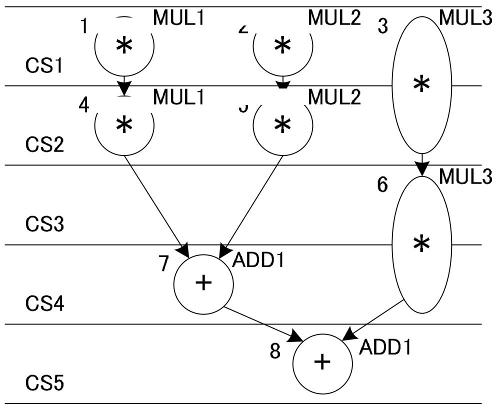


Figure 3.5: An example of scheduling/binding for HDR architecture.

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

Definition 3.1. *The high-level synthesis problem is, for a given CDFG, a clock cycle constraint, a control step 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, and to assign a supply voltage to each huddle so that the given CDFG is executed correctly considering multi-cycle interconnect communications. The objective is to minimize the total energy consumption.*

3.4 The HLS Algorithm

In this section, a new high-level synthesis algorithm targeting HDR architecture is proposed. The algorithm deals with multiple supply voltages and multi-cycle interconnect communication simultaneously.

Generally, high-level synthesis algorithms considering multi-cycle interconnect communication are composed of schedulings, bindings, and floorplannings and classified into the following two types:

Type 1: Schedulings, bindings, and floorplannings are executed a predetermined number of times in a predetermined order.

Type 2: Schedulings, bindings, and floorplannings are executed repeatedly as an iterative refinement flow.

In Type 1, a required time to synthesize a chip can be expected easily since how many times each synthesis step is executed and its execution order are determined. If we know in advance how many times we need to perform each high-level synthesis step as well as its best execution order, Type 1 will be the best choice. MCAS in [8], which is one of the RDR architecture synthesis algorithms, uses an approach based on Type 1 above. Since RDR architectures have uniform-sized islands, inter-island delays are unchanged even if RDR island configurations are changed. Then we can execute a predetermined design flow as in Type 1 above.

In Type 2, several informations such as scheduling results and placement results are fed back to each other since each synthesis step is executed repeatedly as many times as needed. A GDR architecture synthesis algorithm proposed in [32] uses an approach based on Type 2. By iteratively executing scheduling/binding steps and floorplanning steps, a current scheduling/binding step can use interconnection delays obtained in a previous floorplanning step. The shape and size of each module are determined in a scheduling/binding step and a floorplanning step is done using these module informations. Because each synthesis step affects each other, an iterative refinement flow as in Type 2 must be the best choice targeting GDR architectures.

As far as I know, all the existing high-level synthesis approaches based on Type 1 just ignore level converter delays or assume that level converters are inserted between all the two modules [4, 18, 25, 26, 34, 35, 38, 47, 48]. This is because it is very difficult to insert level converters and other required components when they are needed. On the other hand, we can consider level converters and other required components in scheduling and binding according to module configurations determined at the previous iteration in Type 2. It is very natural that we develop an algorithm based on Type 2 when we consider multiple supply voltages in HDR architectures. Totally, Type 2 will be the best choice in HDR architecture synthesis algorithms.

An HDR architecture synthesis algorithm based on Type 2 must have the following four steps in each iteration:

- **scheduling/binding**
- **register/controller synthesis and floorplanning**
- **huddling, and**
- **unhuddling.**

In a scheduling/binding step, each operation node in a CDFG is assigned to a control step and a functional unit considering multiple supply voltages and multi-cycle interconnect communications. In a register/controller synthesis and floorplanning step, registers and controller configuration in each huddle are determined using a scheduling/binding result and every huddle is placed on a chip. In a huddling step, adjacent huddles are merged into a single huddle. In an unhuddling step, a single huddle is partitioned into several huddles.

Now we face a problem when and how many times each of the above four synthesis steps is executed in each iteration (see Fig. 3.6). Assume that we have initial huddle configurations somehow. Then we can execute (i) a scheduling/binding step based on them. After a scheduling/binding step is done, (ii) a register/controller synthesis and floorplanning step must be executed since an operation scheduling to a control step and binding to a function unit may be changed. After that we can merge two or more huddles into a single huddle since floorplanning may be changed. This means that (iii) a huddling step can be done after Step (ii). If several huddles are merged into a single huddle, a register/controller synthesis is needed since each huddle has a single registers and a controller. We need (iv) a register/controller synthesis and floorplanning step again. Finally, we try (v) an unhuddling step and if no huddles are unhuddled, we can finish the loop or we continue Steps (i)–(v) again.

The remaining problem is how to obtain initial huddle configurations. We can assume the following two initial huddle configuration options:

Option 1: As an initial state, we assume a single huddle which contains all the given functional units.

Option 2: As an initial state, we assume several huddles, each of which includes only a single functional unit.

In Option 1, the synthesis flow is roughly based on unhuddling a single huddle into multiple huddles. However, we cannot find out which part in a huddle will cause a multi-cycle commutation since we have only a single huddle or two in an early iteration stage. Moreover, we cannot assign multiple supply voltages to a single huddle since each huddle has its own supply voltage.

In Option 2, the synthesis flow is roughly based on huddling two or more huddles into a single huddle. If two or more huddles are placed close to each other, they should be merged into a single huddle unless they cause interconnection delay errors. We can also deal with multiple supply voltages by considering multiple huddles and assigning an appropriate supply voltage to each of them.

Based on the above discussion, we can say that Option 2 is the best choice as the initial huddle configurations. Overall, we can summarize that Fig. 3.6 shows the best synthesis flow targeting HDR architectures. In initial huddling, initial huddle configuration and placement are determined by given functional units. If p functional units are given as input, we prepare just p huddles in which each functional unit is assigned to each huddle. We merged huddles by huddling during Steps (i)–(v) iteratively. When no huddles are partitioned in Step (v) (in other words, no timing violations occur in Step (iv)), the iteration is finished.[1] In the rest of this section, each process in Fig. 3.6 will be proposed. Note that only DFG is used for simplicity as a motivated example but we can deal with CDFG similarly. In the same way, adders and multipliers with fixed bit width as functional units are used for simplicity.

3.4.1 Initial huddling

In initial huddling, initial huddle configuration and placement are determined by given functional units. If p functional units are given as input, we prepare just p huddles in which each functional unit is assigned to each huddle and the supply voltage v_h is assigned to each huddle. All the huddles are overlapped with each other where we can ignore interconnection delays between huddles here. $F(h_i)$, $V(h_i)$, and $Dist(h_i, h_j)$ are set to be:

$$\begin{aligned} F(h_i) &= \{f_i\}, && (1 \le i \le p) \\ V(h_i) &= v_h, && (1 \le i \le p) \\ Dist(h_i, h_j) &= 0. && (1 \le i, j \le p) \end{aligned}$$

After initial huddling, we will start the first iteration.

[1]It is not guaranteed that the proposed algorithm always generates converged results even when there exists a feasible solution, but the algorithm has converged in 2–8 iterations in the experimental results in Section 3.5. Note that other DR architecture synthesis algorithms also have the similar convergence problem above.

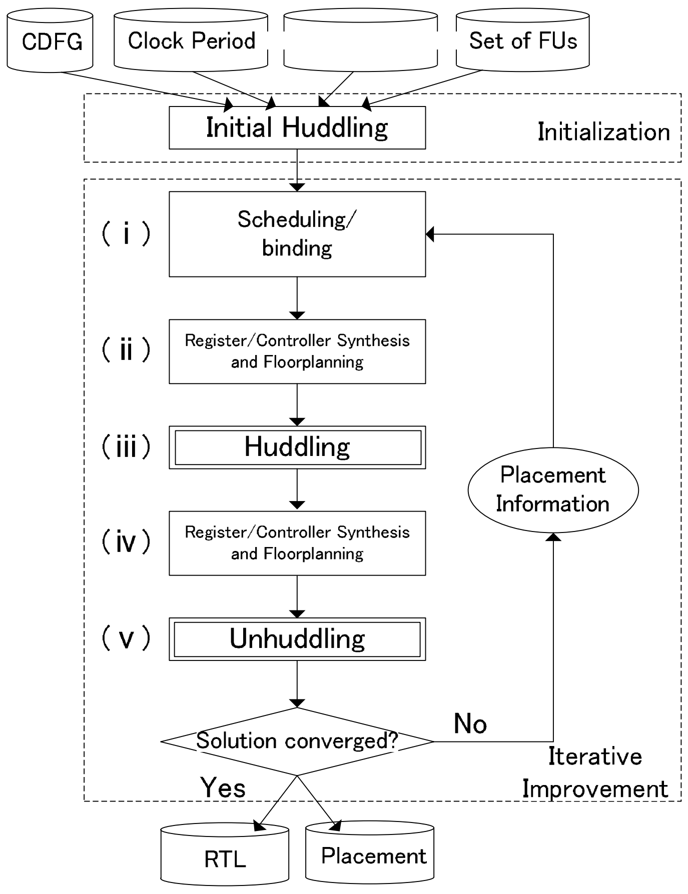


Figure 3.6: Energy-efficient high-level synthesis algorithm targeting HDR architectures.

3.4.2 Scheduling/binding

When the supply voltage assigned to operations is changed, the execution timing and the energy consumption of the operation are also changed. When low supply voltage is assigned to an operation, its execution time will increase and its energy consumption will decrease. If an operation execution timing is changed, we may change the operation scheduling and/or operation binding. The proposed algorithm has five steps (i)–(v) but only Step (i) determines operation execution timings. It is very natural that we assign supply voltages to huddles in the scheduling/binding step and the other steps will be carried out based on supply voltages determined in Step (i).

Then the scheduling/binding problem is, for given a CDFG $G(N, E)$, a clock period constraint T_{clk}, a control step constraint S_{max}, a set of functional units, huddle configuration, an initial supply voltage assigned to each huddle, and huddle placement, to find scheduling and functional unit binding of every node in a given CDFG and to determine supply voltages assigned to given huddles so as to minimize the total energy consumption meeting clock period constraint and control step constraint. Note that, interconnection delays are ignored in the first iteration since floorplanning is not carried out and we assume that all the huddles are overlapped with each other.

The scheduling/binding is composed of the three phases: initial phase, voltage-increasing phase, and voltage-decreasing phase. In the initial phase, scheduling and binding are executed according to the previous huddle placement and voltages. Since operation binding may be changed, its voltages may be changed in this phase but huddle voltages are not changed. Voltage-increasing phase is executed when the initial phase result does not satisfy the control step constraint and huddle voltages are increased so as to satisfy the control step constraint. Voltage-decreasing phase decreases huddle voltages so as to minimize total energy consumption while meeting the control step constraint.

In order to minimize the energy consumption so as to satisfy control step constraint through the voltage-increasing phase and the voltage-decreasing phase, we design a priority $P_s(h_j)$ for a huddle h_i. We will change the supply voltage of h_i based on its priority. The priority $P_s(h_j)$ is calculated by:

$$P_s(h_j) = \sum_{f_i \in F(h_j)} E(f_i)/D(f_i). \tag{3.5}$$

According to [26], $P_s(h_j)$ expresses the energy-efficient effect that is caused by assigning the voltage to h_j. If low voltage can be assigned to h_j that has high $P_s(h_j)$, we can gain farther reduction of overall energy consumption.

Initial phase is executed as a first step of scheduling/binding according to huddle placement and voltages obtained by the previous iteration. Fig. 3.7 shows the initial phase. Basically, we use data-transfer-table based scheduling [31]. If the initial phase result here does not satisfy the control step constraint, we will execute the voltage-increasing phase. Otherwise, we will execute the voltage-decreasing phase.

Voltage-increasing phase is executed when the initial phase result does not satisfy the control step constraint. Th voltage-increasing phase will increase huddle voltages and satisfy the control step constraint. Fig. 3.8 shows the voltage-increasing phase. We first try to increase the huddle voltage from v_l to v_m to satisfy the control step constraint. At that time we pick up the huddle h_j with the voltage v_l whose priority $P_s(h_j)$ is the smallest first. This is because, even if the supply voltage of h_j is increased to from v_l to v_m, we expect that overall energy consumption is as small as possible satisfying the control step constraint. If the control step

Initial Phase.

1. Calculate data transfer table $DT(f_i, h_k)$ for each functional unit f_i and each huddle h_k.
2. Perform scheduling/binding based on the data transfer table $DT(f_i, h_k)$ [31].
3. If the result satisfies the control step constraint, perform (a)–(f) so as to minimize energy consumption:
   (a) If there are multipliers with the voltage v_l and v_m,
      i. Select a multiplication node n which are executed with the voltage v_m.
      ii. Assume that n is executed with the voltage v_l (not v_m), and perform scheduling/binding based on $DT(f_i, h_k)$ [31] without changing any other operation voltages.
      iii. If the result satisfies the control step constraint, we accept the result. Otherwise, we execute n with the original voltage v_m.
      iv. Repeat the above steps until we try all the multiplication nodes executed with v_m.
   (b) If there are multipliers with the voltage v_l and v_h, perform the same steps above for multiplication node executed with the voltage v_h.
   (c) If there are multipliers with the voltage v_m and v_h, perform the same steps above for multiplication node executed with the voltage v_h.
   (d) If there are adders with the voltage v_l and v_m, perform the same steps above for addition node executed with the voltage v_m.
   (e) If there are adders with the voltage v_l and v_h, perform the same steps above for addition node executed with the voltage v_h.
   (f) If there are adders with the voltage v_m and v_h, perform the same steps above for addition node executed with the voltage v_h.

Figure 3.7: Initial phase in scheduling/binding.

constraint is not satisfied when we increase all the huddle voltages from v_l to v_m, we try to increase the huddle voltage from v_m to v_h in a similar way.

Voltage-decreasing phase decreases huddle voltages so as to minimize total energy consumption while meeting the control step constraint. Fig. 3.9 shows the voltage-decreasing phase. We first try to decrease the huddle voltage from v_h to v_m while meeting the control step constraint. At that time we pick up the huddle h_j with the voltage v_h whose priority $P_s(h_j)$ is the largest first. After that, we try to decrease the huddle voltage from v_m to v_l in a similar way.

Example 3.3. *Let us consider a DFG as depicted in Fig. 3.10(a). Assume that the clock cycle constraint of* $T_{clk} = 3\,\mathrm{ns}$ *and the control step constraint* $S_{max} = 8$ *are given. Tables 3.1 and 3.2 summarize functional unit and level converter specifications. Huddle configurations of Fig. 3.10(b) are also given and we assume*

Voltage-increasing phase

1. Pick up a huddle h_j with the smallest priority $P_s(h_j)$ among the huddles executed with the voltage v_l. Change its voltage from v_l to v_m.
2. Perform the initial phase again (Fig. 3.7).
3. If the result satisfies the control step constraint, finish.
4. Repeat the above steps 1–3 until all the huddles with the voltage v_l are tried.
5. Pick up a huddle h_j with the smallest priority $P_s(h_j)$ among the huddles executed with the voltage v_m. Change its voltage from v_m to v_h.
6. Perform the initial phase again (Fig. 3.7).
7. If the result satisfies the control step constraint, finish.
8. Repeat the above steps 5–7 until all the huddles with the voltage v_m are tried.

Figure 3.8: Voltage-increasing phase in scheduling/binding.

that the interconnection delays between the three huddles as $D_w(Dist(h_1, h_3)) = D_w(Dist(h_1, h_2)) = D_w(Dist(h_2, h_3)) = 1\,\text{ns}$. *Register delays are given by* $D_{reg}(h_1) = D_{reg}(h_2) = D_{reg}(h_3) = 0.5\,\text{ns}$.

At the initial phase, a data transfer table $DT(f_i, h_k)$ *is constructed first. F . 3.10(c) shows the constructed data transfer table* $DT(f_i, h_k)$ *and the input DFG is scheduled as in Fig. 3.10(a). Since Fig. 3.10(a) does not satisfy the control step constraint, we execute the voltage-increasing phase next.*

In the voltage-increasing phase, we pick up the huddle h_3 *in Fig. 3.10(b) with the voltage* v_l *having the smallest priority. Fig. 3.11(b) shows the result where the huddle voltage* $V(h_3)$ *is changed from* v_l *to* v_m. *After that,* $DT(f_i, h_k)$ *is re-constructed and the initial phase is executed again. In this case, we have* $DT(f_i, h_k)$ *as in Fig. 3.11(c) and we have a scheduling result of Fig. 3.11(a). Since Fig. 3.11(a) satisfies the control step constraint, we execute the voltage-decreasing phase next.*

In the voltage-decreasing phase, we first pick up a huddle with the voltage v_h *with the largest priority but there exists no huddles with the voltage* v_h *in Fig. 3.11(b). Then we pick up the huddle* h_3 *with the voltage* v_m *having the largest priority and its huddle voltage* $V(h_3)$ *is changed from* v_m *to* v_l. *In this case, we obtain a scheduling result of Fig. 3.10(a), which is the same result of the initial phase. However, since Fig. 3.10(a) does not satisfy the control step constraint,* $V(h_3)$ *is returned to the original voltage* v_m.

In a similar way, the voltage-decreasing phase continues. However, the result

Voltage-decreasing phase

1. Pick up a huddle h_j with the largest priority $P_s(h_j)$ among the huddles executed with the voltage v_h. Change its voltage from v_h to v_m.
2. Perform the initial phase again (Fig. 3.7).
3. If the result does not satisfy the control step constraint, we assign the original voltage v_h to h_j.
4. Repeat the above steps 1–3 until all the huddles with the voltage v_h are tried.
5. Pick up a huddle h_j with the largest priority $P_s(h_j)$ among the huddles executed with the voltage v_m. Change its voltage from v_m to v_l.
6. Perform the initial phase again (Fig. 3.7).
7. If the result does not satisfy the control step constraint, we assign the original voltage v_m to h_j.
8. Repeat the above steps 5–7 until all the huddles with the voltage v_m are tried.

Figure 3.9: Voltage-decreasing phase algorithm.

satisfying the control step constraint can not be obtained. So we can finally have a result of Fig. 3.11(a) satisfying the control step constraint with the smallest energy consumption. □

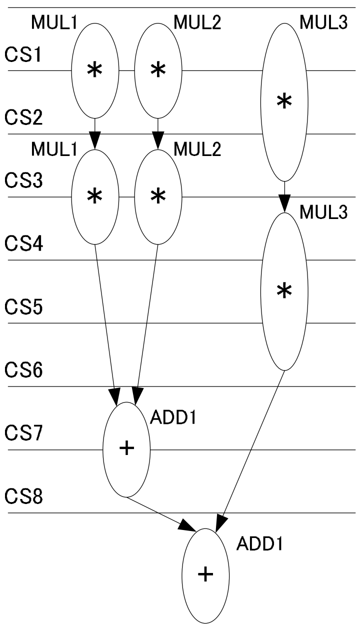
3.4.3 Register/controller synthesis and floorplanning

In the register/controller synthesis and floorplanning step, register and controller configuration in each huddle is determined according to the result of a scheduling/binding step and then huddle placement is optimized. The same algorithm with GDR architectures [32] is applied to the register/controller synthesis for HDR architectures. Since we can determine which components are assigned to each huddle, we can also determine the total area of each huddle.

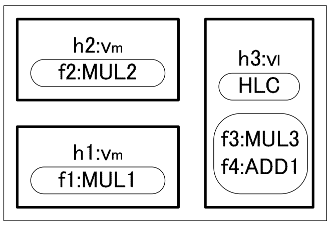
Huddle placement as well as its height and width is optimized by using a simulated annealing (SA) strategy based on a sequence-pair representation [28]. In the floorplanning, power network resources are considered as in [22]. In SA optimization, its cost function *cost* is expressed by

$$cost = \frac{A_{BB}}{A_{total}} + \alpha\frac{V}{T_{clock}} + \beta\frac{W}{W_{MAX}} + \gamma\frac{A_{PNR}}{A_{total}} \tag{3.6}$$

where A_{BB} is the rectangle area which includes all the huddles (dead space may be included), A_{total} is the sum of huddles' area (dead space is not included), T_{clock}



(a) DFG.



(b) Placement information.

	h1	h2	h3
f1	0	1	2
f2	1	0	2
f3	2	2	0
f4	2	2	0

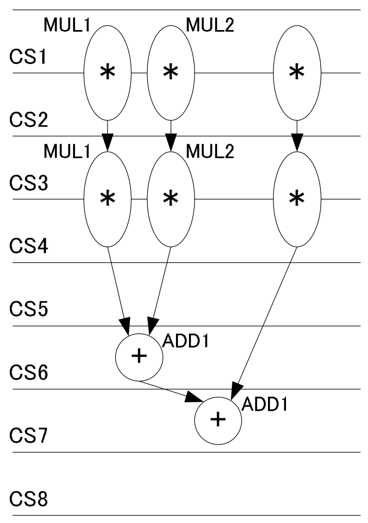
(c) Data transfer table.

Figure 3.10: Inputs of the scheduling/binding.

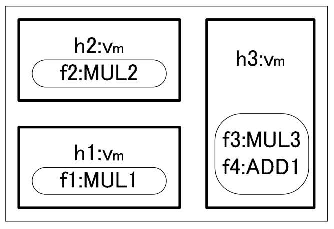
is the clock period constraint, V is the sum of the excess data transfer time which violates $DT(f_i, h_k)$ of each f_i and h_k, W is the wire length, W_{MAX} is the max wire length calculated by (rectangle area's height + width)×the number of wires, and A_{PNR} is the sum of the rectangle area of the respective voltages which includes all the huddles where the voltages are assigned (dead space may be included). α, β and γ are parameters.

The initial solution of floorplan at each iteration is the floorplan solution represented by its sequence-pair of the previous result so that the entire iteration in Fig. 3.6 can converge gradually. Initial temperature T_i in floorplan at the i-th iteration of the synthesis flow is computed by

$$T_{i+1} = KT_i \tag{3.7}$$



(a) DFG.



(b) Placement information.

	h1	h2	h3
f1	0	1	1
f2	1	0	1
f3	1	1	0
f4	1	1	0

(c) Data transfer table.

Figure 3.11: Outputs of the scheduling/binding.

where K is also a parameter and set to be $K < 1$.[2]

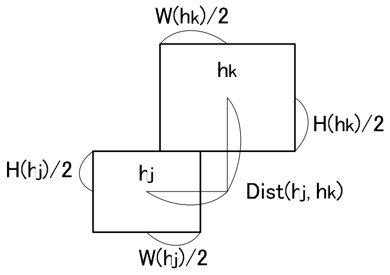
Note that Step (ii) of register/controller synthesis and floorplanning and Step (iv) of register/controller synthesis and floorplanning in Fig. 3.6 are completely the same steps.

3.4.4 Huddling

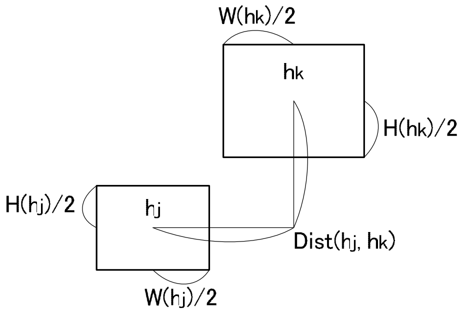
In huddling, we merge adjacent huddles into a single huddle based on the floorplan result. Since the floorplanning cost is calculated by Eqn. (3.6), huddles that should be merged into a single huddle must be placed close to each other.

In order to determine huddles that should be merged, adjacency $Adj(h_j, h_k)$ for

[2]In the experiments, $\alpha = 100$, $\beta = 1$, $\gamma = 0.5$ and $K = 0.9$ were set.



(a) Adjacent huddles h_j and h_k.



(b) Not adjacent huddles h_j and h_k.

Figure 3.12: An example of adjacency $\mathit{Adj}(h_j, h_k)$.

huddles h_j and h_k $(j \neq k)$ is defined as:

$$\mathit{Adj}(h_j, h_k) = \left[\frac{H(h_j)}{2} + \frac{H(h_k)}{2}\right] + \left[\frac{W(h_j)}{2} + \frac{W(h_k)}{2}\right] - \mathit{Dist}(h_j, h_k). \tag{3.8}$$

$\mathit{Adj}(h_j, h_k)$ will be positive when h_j and h_k are adjacent and $\mathit{Adj}(h_j, h_k)$ will be negative when h_j and h_k are placed far away.[3] Fig. 3.12(a) shows the case that the two huddles are adjacent and $\mathit{Adj}(h_j, h_k) \geq 0$. Fig. 3.12(b) shows the case that the two huddles are not adjacent and $\mathit{Adj}(h_j, h_k) < 0$. $HC(h_j, h_k)$ shows the number of inter-huddle connections between huddles h_j and h_k where $HC(h_j, h_k) \geq 0$.

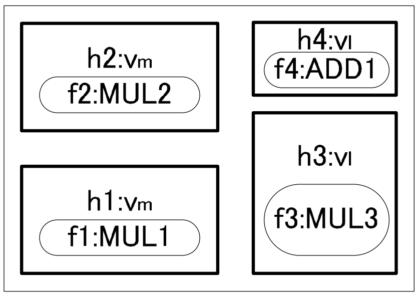
Then we can define the priority $P_h(h_j, h_k)$ which shows whether huddle h_j and h_k should be merged or not:

$$P_h(h_j, h_k) = \mathit{Adj}(h_j, h_k) \cdot HC(h_j, h_k). \tag{3.9}$$

In a similar way, $P_h(h_j, h_k)$ will be positive when h_j and h_k are adjacent and $P_h(h_j, h_k)$ will be negative when h_j and h_k are placed far away.We first pick a pair of huddles whose P_h value is the largest and check whether these huddles satisfy the merging condition. When they satisfy the merging condition, they are merged

[3] $\mathit{Adj}(h_j, h_k)$ can be positive even if the two huddles h_j and h_k are not adjacent, but we can say that the two huddles are close enough if $\mathit{Adj}(h_j, h_k)$ is positive. This is because of the following reason:

The merging priority $P_h(h_j, h_k)$ in Eqn. (3.9) should represent the amount $HC(h_j, h_k)$ of data transfers in each combination of huddles. However, two huddles which are placed far away should not be merged into a single huddle since a floorplanning result indicates some optimal situation. Therefore we also require the criterion how much close the two huddles are placed. Thus we define $\mathit{Adj}(h_j, h_k)$ as in Eqn. (3.8). In fact, the huddling works well in the experiments in Section 3.5.

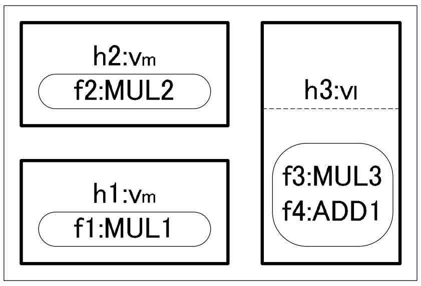


(a) Placement.

	h1	h2	h3	h4
h1		15	10	−10
h2	15		5	10
h3	10	5		15
h4	−10	10	15	

(b) Priority $P_h(h_j, h_k)$ between the two huddles

Figure 3.13: Inputs of huddling.



(a) Placement.

	h1	h2	h3
h1		15	10
h2	15		10
h3	10	10	

(b) Priority $P_h(h_j, h_k)$ between the two huddles

Figure 3.14: Outputs of huddling.

into a single huddle. The merging condition here is defined by

$$\begin{cases} V(h_j) = V(h_k) \text{ and} \\ \\ h_j \text{ and } h_k \text{ satisfy the huddle size constraint.} \end{cases} \tag{3.10}$$

We continue to find a pair of huddles that satisfy the merging condition until no pair of huddles satisfies the merging condition.

In the huddling, we do not consider overlapping of existing huddles and merged huddles and all pairs of huddles satisfying the merging condition are merged. This overlapping will be resolved at Step (iv) of register/controller synthesis and floorplanning. By introducing this approach, we can have as small number of huddles as possible and will have a floorplanning result consistent with Step (i) of scheduling/binding.

Example 3.4. *Fig. 3.13 shows an example of huddling. The huddle pair of h_3 and h_4 are picked up since they have the maximum priority $P_h(h_3, h_4) = 15$. Since*

$V(h_3) = V(h_4) = v_l$ and they also satisfy the huddle size constraint, they satisfy the merging condition. Then h_3 and h_4 are merged into a single huddle h_3.

We check other pairs of huddles, but h_1 and h_2 do not satisfy the huddle size constraint and other pairs of huddles have different supply voltages.

Overall, we can finally have a new huddle configuration as in Fig. 3.14(a). □

3.4.5 Unhuddling

In Section 3.4.4, we have proposed a huddling step which merges several huddles into a single huddle, but a synthesis solution may fall into a local minimum if we only deal with huddling. We need an unhuddling step which partitions a single huddle into several huddles.

In unhuddling, we also utilize huddle placement information. Let $DT_s(f_i, h_k)$ and $DT_f(f_i, h_k)$ be data transfer tables for a functional unit f_i and a huddle h_k, just after (i) scheduling/binding step in the current iteration and just after (iv) register/controller synthesis and floorplanning step in the current iteration, respectively. Then we check whether the following equation holds true or not:

$$DT_s(f_i, h_k) < DT_f(f_i, h_k). \tag{3.11}$$

If Eqn. (3.11) holds, a data transfer delay from f_i to h_k may violate the given clock period constraint. In this case, f_i cannot be assigned to huddle $h_j(= Hud(f_i))$. We eliminate f_i from h_j, construct a new huddle h_l, and assign f_i to the new huddle h_l. h_l is placed so as to overlap h_k. In (ii) register/controller synthesis and floorplanning step of the next iteration, registers and controller for the new huddle h_l will be constructed and the overlap of huddles will be resolved.

If no pair of huddles satisfy Eqn. (3.11), all the data transfer times based on the current floorplanning do not exceed the times which are calculated in the scheduling/binding. Then we will finish the the iterative improvement loop.

3.5 Experimental Results

In this section, the circuit models are described and the proposed algorithm are evaluated.

The interconnection delay model

In this dissertation, the interconnection delays are assumed to be proportional to the square of the wiring length and an interconnection delay was set to be 1 ns when wiring length is 250 μm. The interconnection delay model is the same model in [32]. In [32], they use the values in ITRS '05 [13] and obtain the ratio between gate delay and interconnection delay in CMOS 45nm technology. Then they apply this ratio to CMOS 90nm technology and obtain an interconnection delay here. This is because:

- They have only CMOS 90nm technology library, but they do not have CMOS 45nm technology.
- Multi-cycle interconnect communication is required in technology nodes finer than 65nm technology, say 45nm technology.

In ITRS '05 [13], we can see:

Interconnection delay of CMOS 90nm technology: 1 ns per 2272 μm.

Gate delay: Roughly saying, the gate delay of CMOS 45nm technology is 0.492 times smaller than that of CMOS 90nm technology.

Interconnection delay: Similarly, the interconnection delay of CMOS 45nm technology is 4.51 times larger than that of CMOS 90nm technology.

Then the converted interconnection delay in CMOS 90nm technology will become 2272/(4.51/0.492) = 248.31, meaning 1ns per 248.31 μm. Therefore in [32], the interconnection delay is estimated as 1ns per 250 μm.

In [8], interconnection delay is directly proportional to the wire length and about 1ns per 50mm. This is because they are based on the interconnection delay model on FPGAs. They are based on 70nm technology FPGA and they assumed that optimal buffer insertion and wire sizing are performed. They have been described "*five clock cycles are still needed to go from corner-to-corner for the predicted die of 28.3mm$\times$28.3mm in the 70-nm technology generation, assuming a 5.63-GHz clock*". Then, we consider interconnection delay without optimal buffer insertion in their experimental environment. The interconnection delay of CMOS 68nm technology requires 890ps when wiring length is 1mm according to ITRS '07 [14]. By using this value, it takes approximately 2851168.4ps to transfer 28.3mm $\times$ 28.3mm (we also assume here that interconnection delay is proportional to the square of wire length). In other words, it requires 16052 cycles in the case of 5.63GHz clock (the buffers have to be inserted to prevent this situation).

In contrast, the distance which can be communicated in 1 clock cycle is about 437μm. These things lead to two conclusions:

- First, very large-scale circuits requires optimized buffer insertions in order to optimize interconnection delays, as in [8].

- Second, relatively small/middle-scale circuits without buffers require multi-cycle interconnect communication. In this case, distributed-register architectures which realize multi-cycle interconnect communication can be the best answer.

In this chapter, the proposed algorithm is applied to up to $129600\,\mu\text{m}^2(360\,\mu\text{m} \times 360\,\mu\text{m})$ circuits in Section 3.5. Then buffer insertion is not required.

Note that, the forecast of interconnection delays in ITRS '05 [13] is smaller than the measured values in ITRS '07 [14] and ITRS '09 [15]. In recent LSI design, some wiring techniques, such as the copper wire and the low-k structure for wire, have been proposed, but the trend of interconnection delay will continue over successive years. Therefore the converted interconnection delay above is not so exaggerated and the interconnection delay model is used in this dissertation. This interconnection delay model is also used in the latter chapter.

The level converter model

In Chapter 3 and Chapter 4, the information of the level converter is obtained from [40]. Since the information of level converters in [40] is based on the CMOS 65 nm technology, the circuit information is converted to that of the CMOS 90 nm technology. In Chapter 3 and Chapter 4, level converters are inserted when data is transfered not only from lower voltages to higher voltages but also from higher voltages to lower voltages.

The results

The proposed algorithm have been implemented in C++. The algorithm has been applied to DCT (a discrete cosine transform algorithm for 8×8 pixels, 48 nodes), Jacobi (Jacobi method to solve linear equations with four unknown variables, 48 nodes), EWF3 (three elliptic wave filters are serially connected, 102 nodes), and FIR filter (a seventh order finite impulse response filter, 75 nodes)[4]. Table 3.3 shows the functional unit specification and Table 3.4 shows the level converter specification [40][5]. All the functional units were assumed to have a bit width of 16, and their specifications were obtained by synthesizing them beforehand based on the CMOS 90 nm technology. Controllers were synthesized by Synopsys Design

[4]The benchmarks use the condition vector (CV) [44] and are described by CoDaMa [20] which are written in XML. The benchmarks are used in [31, 32, 39]. COPY and PARKER in the letter chapter are the same.

[5]Since the information of level converters in [40] is based on the CMOS 65 nm technology, the circuit information is converted to that of the CMOS 90 nm technology.

Compiler in each iteration. The interconnection delays were assumed to be proportional to the square of the wiring length and an interconnection delay was set to be 1 ns when wiring length is 250μm [32].Energy consumption is obtained using Synopsys Design Compiler.

The proposed algorithm targeting HDR architectures with multiple supply voltages ("MHDR" in Table 3.5) is compared to a GDR architecture synthesis algorithm [32] ("GDR" in Table 3.5), MCAS for RDR architectures [8] ("RDR" in Table 3.5), and the proposed algorithm targeting HDR architecture with a single supply voltage ("HDR" in Table 3.5). The proposed algorithm is further compared with the following strategy: the existing multiple supply voltage aware scheduling [48] is first performed; based on this voltage assignment to each operation, MCAS for RDR architecture is performed (" [48] + RDR" in Table 3.5), and the proposed algorithm targeting HDR architecture is performed (" [48] + HDR" in Table 3.5). The clock period constraint was given to be 2.5 ns in all the experiments.

Table 3.5 summarizes the experimental results. In Table 3.5 "CS constraints" shows the control step constraint. "Control steps" shows the number of required control steps after synthesizing each circuit. "Area" and "Rectangular area" in Table 3.5 represent the sum of module/huddle area and the minimum rectangle area including all of them. "Dynamic energy" and "Leak energy" represent dynamic energy consumption and leakage energy consumption. "All Energy" shows the sum of "Dynamic energy" and "Leak energy". "Iterations" shows the number of iterations requred by each algorithm. "CPU Time" shows CPU time to synthesize each circuit.

The experimental results show that the smallest area is "GDR", HDRs("HDR", " [48] + HDR", "MHDR"), and RDRs("RDR", " [48] + RDR") in that order. However, "GDR" areas can be sometimes larger than "HDR" areas. This is because of the following reason: "GDR" has shared register groups but, since its synthesis flow is too complicated as pointed out in Section 3.2, several registers cannot be shared into any shared register group and become local registers in order to meet the timing constraints. On the other hand, the "HDR" has a strucutre of huddles and all the regsters in each huddle are shared into the expected shared registers.

Between the areas considering single supply voltage and those considering multiple supply voltages, the latter will be larger in most cases. This is because the level converter area must be added and the results considering multiple supply voltages decrease register sharing. However, "MHDR" areas can be sometimes become smaller than "HDR" areas. This is just because of the proposed algorithm cannot always have an optimal (or semi-optimal) result. Since the proposed algorithm is based on an iterative improvement flow, it sometimes fall into an local optimum. In the case of EWF3 using HDR, it is just the case.

Table 3.3: Functional unit specification.

Functional Unit	Area [μm^2]	Delay [ns]	Dynamic energy [fJ]	Leak power [μW]
Adder (1.2 V)	386	0.75	92	3.9
Adder (1.0 V)	386	1.22	64	3.2
Adder (0.8 V)	386	2.71	41	2.6
Subtractor (1.2 V)	417	0.78	97	4.2
Subtractor (1.0 V)	417	1.27	67	3.5
Subtractor (0.8 V)	417	2.82	43	2.8
Multiplier (1.2 V)	2161	1.65	1135	19.8
Multiplier (1.0 V)	2161	2.70	788	16.5
Multiplier (0.8 V)	2161	6.00	504	13.2
Divider (1.2 V)	6066	6.25	2306	837.6
Divider (1.0 V)	6066	10.21	1601	698.0
Divider (0.8 V)	6066	22.69	1234	524.0

The experimental results show that the dynamic energy consumption of "MHDR" is reduced by a maximum of 48.2% and an average of 25.2% compared with the other algorithms. All energy consumption of "MHDR" is also reduced by a maximum of 48.1% and an average of 22.4% compared with the other algorithms. In "GDR", "RDR", and "HDR", multiple supply voltages can not been considered. Since "MHDR" can assign lower voltages to non-critical pathes, "MHDR" achieved maximally 48.1% energy reduction. The leakage energy consumption of "MHDR" is reduced by a maximum of 60.3% , but is increased by an average 9.4% compared with the other algorithms. This is because level converters increase the leakage energy, but the overall energy consumpation is much reduced compared with other algorithms.

Note that the objectives of synthesis algorithms of "GDR", "RDR" and "HDR" are to minimize the required control steps. Actually, their control steps are shorter than the control step constraints. All the energies in Table 3.5 are evaluated within the required control steps. These results can be fairly compared to results obtained by " [48]+RDR", " [48]+HDR" and "MHDR". The number of iterations in "MHDR" is up to three and we can have the CPU time comparable to the GDR synthesis algorithm.

Table 3.4: Level converters specification [40].

V_{in} - V_{out}	Area [μm^2]	Delay [ns]	Dynamic energy [fJ]	Leak power [μW]
1.2 V - 1.0 V	113	0.17	83	49.1
1.2 V - 0.8 V	113	0.22	71	32.3
1.0 V - 1.2 V	113	0.17	76	45.0
1.0 V - 0.8 V	113	0.30	55	18.3
0.8 V - 1.2 V	113	0.22	86	39.1
0.8 V - 1.0 V	113	0.30	55	18.3

3.6 Conclusion

In this chapter, I proposed huddle-based distributed register architecture (HDR architecture) for multi-cycle interconnect communications and a new energy-efficient high-level synthesis algorithm targeting HDR architecture. The proposed algorithm reduced energy consumption by a maximum of 48.1% and by an average of 22.4% compared with the conventional algorithms.

Table 3.5: Experimental results

App.	FUs	S_{max}	Architechture	CS	Area	Rectangular	Dynamic	Leak	All	i	CPU time
					[μm^2]	area [μm^2]	[pJ]	[pJ]	[pJ]		[sec]
ewf3	Add×3	50	GDR	43	47792	55250	655.83	57.64	713.47	7	362.48
	Mul×2		RDR	44	69530	69530	764.33	85.06	849.39	1	56.21
			HDR	43	53926	59706	720.82	79.38	800.21	6	1169.40
			[48] + RDR	50	109350	109350	570.41	84.57	654.98	1	127.98
			[48] + HDR	50	57118	60882	577.89	108.90	686.80	8	1815.60
			MHDR	50	44049	48208	520.20	91.07	611.27	2	487.20
fir	Add×3	35	GDR	30	36840	48278	429.16	38.35	467.51	24	1212.06
	Mul×3		RDR	29	81920	81920	360.01	105.18	465.19	1	75.78
			HDR	30	28493	32643	344.88	30.75	375.62	2	353.15
			[48] + RDR	35	115200	115200	507.73	64.94	572.67	1	174.33
			[48] + HDR	35	51795	59220	371.74	75.42	447.17	2	484.67
			MHDR	35	40231	49580	284.64	44.63	329.27	2	484.10
fir	Add×4	30	GDR	30	39407	42593	473.44	40.34	513.78	24	1314.37
	Mul×4		RDR	29	82816	82816	335.95	123.57	459.52	1	75.59
			HDR	30	34967	41087	315.10	38.79	353.89	5	701.88
			[48] + RDR	30	129600	129600	366.16	43.22	409.39	1	190.94
			[48] + HDR	30	57672	66316	475.25	94.20	569.46	2	352.27
			MHDR	30	48011	59175	246.41	49.09	295.49	2	576.32
jacobi	Add×2	20	GDR	19	28026	33660	273.75	60.70	334.45	8	644.76
	Sub×1		RDR	20	57600	57600	224.93	94.94	319.87	1	138.06
	Mul×2		HDR	19	32031	34686	201.13	91.78	292.91	2	288.77
	Div×2		[48] + RDR	20	115200	115200	224.32	119.43	343.74	1	144.62
			[48] + HDR	20	35124	38340	163.21	100.32	263.52	2	448.74
			MHDR	20	36581	42210	163.56	93.41	256.98	2	447.97
dct	Add×4	10	GDR	8	53864	58378	208.48	11.00	219.48	24	1378.30
	Mul×4		RDR	9	81476	81476	220.75	13.69	234.45	1	74.29
			HDR	8	55709	58450	196.58	14.66	211.24	2	505.71
			[48] + RDR	10	115200	115200	235.79	52.98	288.76	1	194.40
			[48] + HDR	10	42272	44544	202.21	22.56	224.77	3	746.99
			MHDR	10	50337	69030	169.16	25.53	194.69	3	822.64

Chapter 4

MH^4: Multiple Supply Voltages Aware High-speed and High-efficiency High-level Synthesis Algorithm for HDR architecture

4.1 Introduction

In Chapter 3, a high-level synthesis algorithm targeting huddle-based distributed-register architectures (HDR architectures) has been proposed where energy-efficiency and interconnection delays are considered simultaneously. The algorithm, however, has the two severe problems: (A) the huddle-area and interconnection-delay oscillation during iterations and (B) the insufficient huddle construction methods.

In this chapter, I propose three new techniques, *virtual area estimation*, *virtual area adaptation*, and *floorplanning-directed huddling* to resolve the problems (A) and (B) and then I propose a new $\underline{\text{m}}$ultiple-supply-voltages aware $\underline{\text{h}}$igh-speed and $\underline{\text{h}}$igh-efficiency $\underline{\text{h}}$igh-level synthesis algorithm for $\underline{\text{H}}$DR architecture called MH^4. Experimental results show that the proposed algorithm achieves about 29.1% run-time-saving compared with the conventional methods, and successfully obtains a solution which cannot be obtained by the algorithm proposed in Chapter 3.

4.2 Problem Definition

The target of the proposed high-level synthesis algorithm is the HDR architecture (Fig. 3.2) composed of huddles, where each huddle includes functional units (HFU), dedicated registers (HLR), level converters (HLC), and dedicated FSM. Rectangular area in each huddle is determined by clock period constraints. Details are shown in Section 3.2.

A control-data flow graph (CDFG) $G(N, E)$ is a directed graph, where a node set N is composed of an operation node set N_o and a branching control node set N_c (start and end nodes of conditional branches), and an edge set E is composed of a data-flow edge set E_d and a control-flow edge set E_c. T_{clk} refers to a clock period constraint and S_{max} refers to a control step constraint.

Let $F = \{f_1, \cdots, f_p\}$ be a set of 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 in $S_f(f_i)$ steps.

Let $H = \{h_1, \cdots, h_q\}$ be a set of huddles in the HDR architecture. Each functional units are bound to any one of the huddles and the binding is defined by a function $Hud : F \to H$. $Hud(f_i)$ is the huddle to which f_i is bound. $F(h_j)$ is a set of functional units which are bound to h_j. $D_{reg}(h_j)$ is a delay of HLRs in h_j.

The three supply voltages, v_l, v_m, and v_h ($v_l < v_m < v_h$), which is assigned to each huddle, are considered. $V(h_j)$ is a supply voltage which are assigned to the huddle h_j. $D_{lc}(v_l, v_m)$ is a delay of a level converter which changes the voltage from v_l to v_m. Likewise, $D_{lc}(v_l, v_h)$, $D_{lc}(v_m, v_l)$, and so on can be defined.

$Slack(f_j)$ is defined by:

$$Slack(f_j) = T_{clk} \cdot S_f(f_i) - D_f(f_i). \tag{4.1}$$

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

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

$$2 \cdot D_w(W(h_j) + H(h_j)) + D_{reg}(h_j) \leq \min_{f_i \in F(h_j)} \{Slack(f_i)\} \tag{4.2}$$

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 whose length is x. In the proposed algorithm, we obtain the value of $(W(h_j) + H(h_j))$ so that it satisfies the huddle size constraint and determine $W(h_j)$ and $H(h_j)$ by using the aspect ratio predefined for each huddle.

Let $Dist(h_j, h_k)$ be the Manhattan distance between the center of huddles h_j and h_k. Then $D_w(Dist(h_j, h_k))$ shows the interconnection delay between them. 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 HLRs in h_k which is defined by:

$$Tr(f_i, h_k) = D_w(Dist(h_j, h_k)) + D_{lc}(V(h_j), V(h_k)) + D_{reg}(h_k). \qquad (4.3)$$

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

$$DT(f_i, h_k) = \begin{cases} 0, & (Slack(f_i) \geq Tr(f_i, h_k)) \\ \lceil Tr(f_i, h_k)/T_{clk} \rceil. & (Slack(f_i) < Tr(f_i, h_k)) \end{cases} \qquad (4.4)$$

In the case of $Slack(f_i) \geq Tr(f_i, h_k)$, the functional unit f_i directly stores its output in the register file of huddle h_k. Thus, the data transfer requires no extra cycles. In the case of $Slack(f_i) < Tr(f_i, h_k)$, the functional unit f_i first stores its output into the local register file in its huddle h_j $(= Hud(f_i))$. In the next cycle, the data transfer from the huddle h_j to the huddle h_k starts. The data transfer requires $\lceil Tr(f_i, h_k)/T_{clk} \rceil$ cycles. Then the data transfer table DT can be defined by a $p \times q$ matrix whose (i, k)-element is expressed by $DT(f_i, h_k)$.

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

Definition 4.1. *The high-level synthesis problem is, for a given CDFG, a clock cycle constraint, a control step 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, and to assign a supply voltage to each huddle. The objective is to minimize the total energy consumption.*

4.3 The MH^4 Algorithm

In Chapter 3, a high-level synthesis algorithm for HDR architectures have been proposed. The scheduling/binding as well as floorplanning are simultaneously optimized by using iterative synthesis flow (Fig. 3.6). The algorithm proposed in Chapter 3 has the two problems:

(A) Huddle-area and interconnection-delay oscillation:

First problem is that huddle areas and interconnection delays may be oscillated during iterations where the same situations might be repeated in the

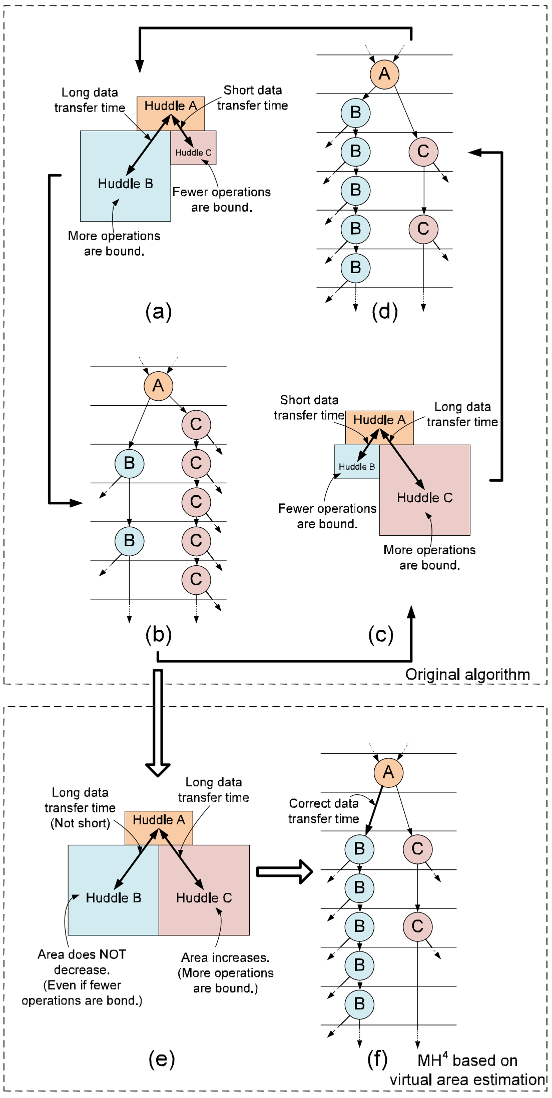

Long data transfer time
Huddle A
Short data transfer time
Huddle C
Huddle B
Fewer operations are bound.
More operations are bound.
(a)
A
B
C
(d)
(b)
Short data transfer time
Long data transfer time
Huddle A
Huddle B
Huddle C
Fewer operations are bound.
More operations are bound.
(c)
Original algorithm
Long data transfer time (Not short)
Long data transfer time
Huddle A
Huddle B
Huddle C
Area does NOT decrease. (Even if fewer operations are bond.)
Area increases. (More operations are bound.)
Correct data transfer time
(e)
(f)
MH^4 based on virtual area estimation

Figure 4.1: Area oscillation example and the virtual area estimation. (a) Floorplanning at the $(i-1)$-th and $(i+1)$-th iterations. (b) Scheduling/binding at the i-th iteration. (c) Floorplanning at the i-th iteration. (d) Scheduling/binding at the $(i+1)$-th iteration. (e) Floorplanning result at the i-th iteration considering virtual area estimation. (f) Scheduling/binding result at the $(i+1)$-th iteration considering virtual area estimation.

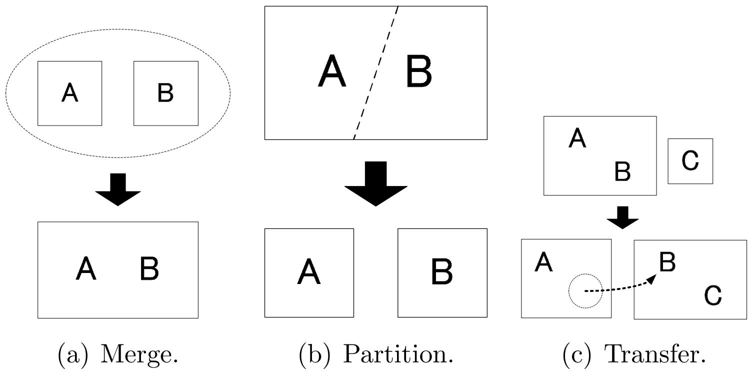


Figure 4.2: The huddle construction methods.

iteration steps. For example, Fig. 4.1(a) shows the floorplanning result at the $(i-1)$-th iteration where too many operations are bound to the huddle B and timing constraints are not satisfied. At the i-th iteration, scheduling/binding is executed based on Fig. 4.1(a). Since the data transfer between the huddles A and C requires shorter time, many operations are bound to C this time and we have the result of Fig. 4.1(b). In this iteration, we have the floorplanning result of Fig. 4.1(c) based on Fig. 4.1(b). C has the larger area in i-th iteration and Fig. 4.1(c) cannot satisfy the timing constraint, either.

In the same way, scheduling/binding is executed based on Fig. 4.1(c) at the $(i+1)$-th iteration and we have Fig. 4.1(d). When we have the floorplanning result based on Fig. 4.1(d), we will go back to the Fig. 4.1(a) and these steps may be repeated.

(B) The insufficient huddle construction methods:

Second problem is the inefficiency of the huddle construction methods. In the original algorithm proposed in Chapter 3, huddles are generated by the two steps *huddling* and *unhuddling*. Since they are not much dependent on each other, we may have poor huddle construction finally. Moreover, huddle construction is composed of *merge*, *partition*, and *transfer* as shown in Fig. 4.2, but the original algorithm only considers merge and partition.

In order to resolve the above problems, I propose three new techniques as follows:

1. **Virtual area estimation:** *A virtual area* is introduced into each huddle. Virtual areas of huddles do not oscillate in the iterations.

2. **Virtual area adaptation:** *Virtual area estimation* above may have some area and interconnection delay overheads. The virtual area adaptation relaxes these overheads as the iterations proceed.

3. **Floorplanning-directed huddling:** In the algorithm proposed in Chapter 3, the two steps, huddling and unhuddling, are executed based on floorplanning results but I embed them into floorplanning as *floorplanning-directed huddling*.

Based on these three techniques, I propose a new multiple-supply-voltages aware high-speed and high-efficiency high-level synthesis algorithm for HDR architectures called MH^4 (Fig. 4.3). MH^4 is mainly composed of the three processes: initial process, iteration process, and adjustment process. In the initial process, initial huddle placement is determined. In the iteration process, scheduling/binding and floorplanning are performed repeatedly based on *virtual area estimation/adaptation*, where huddles are constructed by *floorplanning-directed huddling*. When no timing violations occur in floorplanning-directed huddling, the iteration is finished and we go to the adjustment process. In the adjustment process, real area of each huddle is estimated by the scheduling/binding result obtained in the iteration process. Huddles which no functional unit is assigned to are eliminated in the adjustment process.

Since the processes in MH^4 other than *virtual area estimation*, *virtual area adaptation*, and *floorplanning-directed huddling* are the same as the ones in Section 3.4, I explain here each of the three new techniques.

4.3.1 Virtual area estimation

Problem (A) is mainly caused by huddle area reduction in iterations. If we employ the *maximum area* obtained in each iteration as an estimated huddle area, the huddle area oscillation will not happen and we can expect that the solution will converge very fast without oscillating. The estimated area here is called *virtual area*.

Let $A_{real}(h_j)$ be the original area estimation of the huddle h_j and $A_{virtual}(h_j)$ be the *virtual area* of huddle h_j. $A_{real}(h_j)$ is called a *real area*. Initial value of $A_{virtual}(h_j)$ is set to be the real area obtained by the initial process in MH^4. In each iteration, the $A_{virtual}(h_j)$ is updated if we have larger real area for the huddle h_j. However the $A_{virtual}(h_j)$ is not updated if we have the same or smaller real area for the huddle h_j.

For example, let us consider the case of Figs. 4.1 (a) and (b). After obtaing Fig. 4.1(b), $A_{virtual}(h_C)$ is updated but $A_{virtual}(h_B)$ is not in MH^4 (Fig. 4.1(e)).

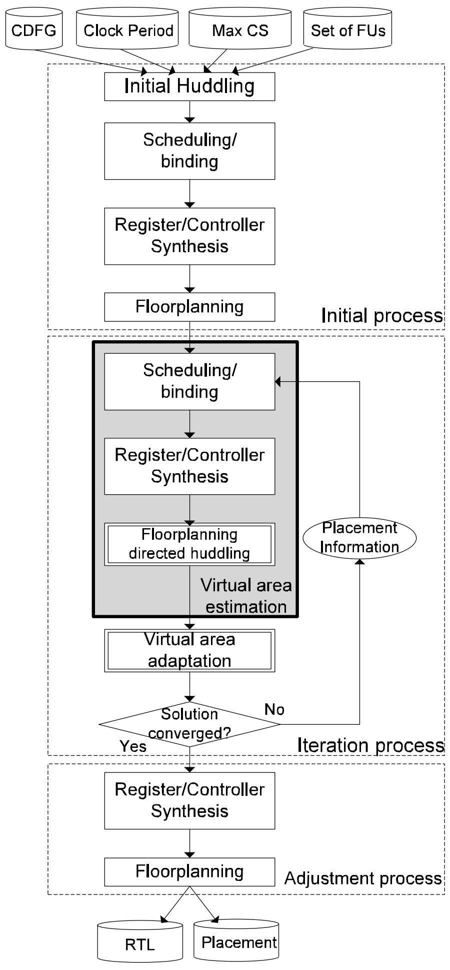


Figure 4.3: The proposed MH^4 algorithm.

When we see the Fig. 4.1(b) and Fig. 4.1(e), we know that the timing violation occurs between the huddles A and C. We finally have the scheduling/binding result of Fig. 4.1(f). After obtaining Fig. 4.1(f), we can also have the same floorplaining result as Fig. 4.1(e) which satisfies the timing constraint. Then we can finish the algorithm iteration.

Overall, the virtual area of the huddle h_j is estimated in the iteration process when its huddle construction is changed as follows:

1. $A_{real}(h_j)$ is calculated by summng up the areas of functional units, registers, a controller, and level converters inside h_j.

2. If $A_{virtual}(h_j) \geq A_{real}(h_j)$, $A_{virtual}(h_j)$ is not updated.

3. If $A_{virtual}(h_j) < A_{real}(h_j)$, we set $A_{virtual}(h_j) = A_{real}(h_j)$.

4.3.2 Virtual area adaptation

Virtual area estimation, however, may increase interconnection delays between huddles as the iterations proceed. To solve this problem, we should decrease the difference between virtual area and real area.

We execute *virtual area adaptation* after floorplanning-directed huddling. Because this step is just before scheduling/binding at the next iteration, we can use virtual areas closer to real areas at the next iteration.

Virtual area adjustment is executed as follows:

1. Let $A_{dif}(h_j) = A_{virtual}(h_j) - A_{real}(h_j)$ be the difference between real area $A_{real}(h_j)$ and virtual area $A_{virtual}(h_j)$ of the huddle h_j.

2. We set $A_{virtual}(h_j) = A_{real}(h_j) + \phi \cdot A_{dif}(h_j)$.

where ϕ is an adaptation parameter. In order to decrease ϕ as the iterations proceed, we set $\phi = \max\{1 - 0.09i, 0\}$ at the i-th iteration.

Note that the virtual area estimations are returned to their real area in the adjustment process.

4.3.3 Floorplanning-directed huddling

Virtual area estimation may cause vacant huddles which no functional unit is assigned to but which has a virtual area. By effectively using vacant huddles, all of the three huddle construction methods *merge*, *partition*, and *transfer* can be represented by just using *transfer* (Figs. 4.4(a) and (b)). This idea can resolve the problem (B).

Huddle construction correlates with floorplanning. It is better for us to integrate huddle construction methods into floorplanning. In floorplaning, huddle placement as well as its height and width is optimized by using a simulated annealing (SA) strategy based on a sequence-pair representation proposed in [28]. In sequence-pair, each module packing is represented by a pair of module name sequences. In the floorplanning-directed huddling, Γ_+ and Γ_- are the huddle name sequences and a pair of Γ_+ and Γ_- represents a placement of huddles. In this step, we consider the four moves as follows:

Move 1: Select two elements and exchange them in Γ_+.

Move 2: Select two elements and exchange them in Γ_+ and Γ_-.

Move 3: Select one element and change its aspect ratio.

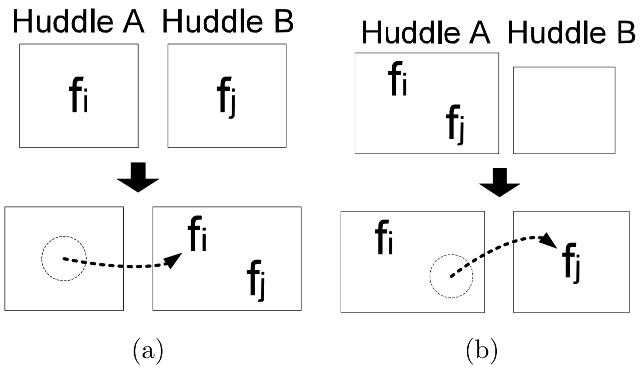


Figure 4.4: (a) Merge using transfer. (b) Partition using transfer.

Move 4: Select functional unit f_i and transfer it from the huddle h_j to the huddle $h_k (\neq h_j)$.

In SA optimization, its cost function *cost* is expressed by

$$cost = \frac{A_{BB}}{A_{total}} + \alpha \frac{V}{T_{clock}} + \beta \frac{W}{W_{MAX}} + \gamma \frac{A_{PNR}}{A_{total}} \tag{4.5}$$

where A_{BB} is the rectangle area which includes all the huddles (dead space may be included), A_{total} is the sum of huddles' area (dead space is not included), T_{clock} is the clock period constraint, V is the sum of the excess data transfer time which violates $DT(f_i, h_k)$ of each f_i and h_k, W is the wire length, W_{MAX} is the max wire length calculated by (rectangle area's height + width)×the number of wires, and A_{PNR} is the sum of the rectangle area of the respective voltages which includes all the huddles where the voltages are assigned (dead space may be included). α, β and γ are parameters. The initial solution of floorplan at each iteration is the floorplan solution represented by its sequence-pair of the previous result. Initial temperature T_i in floorplan at the i-th iteration of the synthesis flow is computed by

$$T_{i+1} = KT_i \tag{4.6}$$

where K is also a parameter and set to be $K < 1$.[1]

4.4 Experimental results

The proposed algorithm has been implemented in C++. The algorithm has been applied to DCT (a discrete cosine transform algorithm for 8×8 pixels, 48 nodes),

[1]In the experiments, $\alpha = 100$, $\beta = 1$, $\gamma = 0.5$ and $K = 0.9$ were set.

EWF3 (three elliptic wave filters are serially connected, 102 nodes) FIR filter (a seventh order finite impulse response filter, 75 nodes), and COPY (provided by a company, 378 nodes, including conditional branches) where COPY is a practical application example. The same functional units and level converters as in Chapter 3 (Table 3.3 and Table 3.4) were used. Selectable voltages were assumed to $v_l = 0.8\,\text{V}$, $v_m = 1.0\,\text{V}$, and $v_h = 1.2\,\text{V}$. Controllers inside huddles were synthesized by Synopsys Design Compiler in each iteration. The interconnection delays were assumed to be a proportion to square of the wiring length and an interconnection delay is set to be 1 ns when wiring length is 250μm. Energy consumption is obtained using Synopsys Design Compiler.

The proposed algorithm ("MH^4" in Table 4.1) has been compared to the GDR architecture synthesis algorithm [32] ("GDR" in Table 4.1), MCAS for RDR architectures [8] ("RDR" in Table 4.1), the HLS algorithm targeting HDR architectures with a single supply voltage ("HDR" in Table 4.1), the HLS algorithm targeting HDR architectures with multiple supply voltages ("MHDR" in Table 4.1) and the proposed algorithm with a single supply voltage ("MH^4 (Single)" in Table 4.1).

The experimental results show that all energy consumption of MH^4 is reduced by a maximum of 30.4% and an average of 18.2% compared with the other algorithms applied to single supply voltage. The CPU time of MH^4 and MH^4 (Single) are reduced by a maximum of 63.9% and an average of 29.1% compared with HDR and MHDR. MH^4 and MH^4 (Single) can obtain a feasible result for COPY which cannot be obtained by HDR and MHDR. All energy consumption in COPY using MH^4 is reduced by a maximum of 57.6% compared with MH^4 (Single). HDR and MHDR can obtain a feasible result for COPY when the clock period constraint was set to be 5.5 ns. In this case, however, all huddles are assigned to 0.8 V. Thus, there is no need to apply multiple supply voltages.

The effectiveness of the adaption parameter ϕ was verified in COPY because the results of them have several iterations and may have large virtual area overheads. The experimental results show that all energy consumption of $\phi = 1 - 0.09i$ is reduced by a maximum of 16.2% and an average of 13.3% compared with that of $\phi = 1.0$ which did not execute virtual area adaptation.

4.5 Conclusion

In this chapter, I propose a multiple-supply-voltages aware high-speed and high-efficiency high-level synthesis algorithm for HDR architectures. The proposed algorithm reduced energy consumption by an average of 18.2% compared with the single-supply-voltage aware algorithms and reduced CPU times by an average of

Table 4.1: Experimental results.

App.	FUs	Clock [ns]	Steps	Architecture and algorithm	ϕ	Rectangular area [μm^2]	All energy [pJ]	CPU time [sec]	Iterations
EWF3	Add×3	1.5	65	GDR	–	42432	476.70	830.43	24
	Mul×2			RDR	–	78400	537.74	105.35	1
				HDR	–	50445	473.72	401.25	2
				MHDR	–	47817	403.12	480.73	2
				MH^4 (Single)	$1-0.09i$	45034	487.60	344.74	2
				MH^4	$1-0.09i$	48399	404.36	357.50	2
FIR	Add×3	1.5	30	GDR	–	22165	198.52	2597.54	24
	Mul×3			RDR	–	99225	241.89	191.02	1
				HDR	–	41856	247.04	635.23	2
				MHDR	–	40040	178.34	725.91	2
				MH^4 (Single)	$1-0.09i$	25992	197.09	523.97	2
				MH^4	$1-0.09i$	37856	171.91	503.32	2
DCT	Add×4	1.5	15	GDR	–	64925	138.07	1338.14	24
	Mul×4			RDR	–	96800	181.25	191.41	1
				HDR	–	60456	164.74	726.11	2
				MHDR	–	65565	129.01	1372.02	4
				MH^4 (Single)	$1-0.09i$	57912	164.74	559.91	2
				MH^4	$1-0.09i$	60060	135.86	495.07	2
COPY	Add×3	1.5	170	HDR	–	–	–	$>$ 1 day	–
	Sub×1			MHDR	–	–	–	$>$ 1 day	–
	Comp×1			MH^4 (Single)	1	338976	8699.10	1900.19	7
	Rshift×2				$1-0.09i$	433246	7805.45	2179.72	6
	AND×1			MH^4	1	402992	3949.53	3618.05	9
	Mul×5				$1-0.09i$	325420	3307.97	2501.39	4
		5.5	90	HDR	–	355320	5714.42	2283.05	4
				MHDR	–	414080	2840.63	2531.08	4
				MH^4 (Single)	$1-0.09i$	374490	5542.46	1091.08	2
				MH^4	$1-0.09i$	336432	2804.23	1822.92	4

29.1% compared with the original algorithm targeting HDR architectures proposed in Chapter 3. The proposed algorithm can successfully obtain optimum high-level synthesis solution which cannot be obtained by the algorithm proposed in Chapter 3.

Chapter 5

SAAV: Dynamic Multiple Supply Voltages Aware High-level Synthesis Algorithm for AVHDR Architecture

5.1 Introduction

In this chapter, I propose *an adaptive voltage huddle-based distributed-register architecture (AVHDR architecture)*, which integrates dynamic multiple supply voltages and floorplanning into HLS. Next, based on the proposed AVHDR architecture, an HLS algorithm is proposed for energy reduction. By using a floorplanning result, the number of non-critical operations can be increased by managing interconnection delay with multi-cycle interconnect communication, and then optimized supply voltages can be assigned for energy reduction. For example, low supply voltages can be assigned to non-critical operations and leakage power can be cut off through power gating. When compared with existing works, on average the energy consumption could be reduced by 43.9%.

5.2 AVHDR Architecture

In this section, the recently proposed distributed-register architectures are briefly reviewed and then the new *adaptive voltage huddle-based distributed-register architecture (AVHDR architecture)* is proposed.

In distributed-register architectures (DR architectures) [8,17,19,31,32], a large circuit is divided into several small partitions. By doing this, 1) the interconnection delay in each partition is assumed to be zero; and 2) the inter-partition data transfer (i.e. the data transfer between partitions) is realized through multi-cycle interconnect communication. Several distributed-register architectures, such as generalized distributed-register architecture (GDR architecture) [32] and regular distributed-register architecture (RDR architecture) [8] have been proposed, but energy-aware HLS algorithms targeting at them have not been proposed. Huddle-based distributed-register architecture (HDR architecture) with the corresponding synthesis algorithm was proposed in Chapter 3 and Chapter 4. HDR architecture can deal with multiple supply voltages (MSV), but this architecture did not consider power gating (PG) and/or dynamic multiple supply voltages (DMSV).

To integrate DMSV and interconnection delay into HLS, a new distributed-register architecture called *adaptive voltage huddle-based distributed-register architecture (AVHDR architecture)* is proposed. AVHDR architecture introduces a non-uniform sized island called a *huddle* in which several functional units are abstracted. The huddle has non-uniform rectangular area under *huddle size constraint*, determined by clock period constraints. We can assume the interconnection delay inside each huddle to be virtually zero and we only need to consider the data transfer time during inter-huddle communication in HLS. In each AVHDR huddle, two types of power supply rails are prepared for applying dynamic multiple supply voltages. One is for *adaptive voltage logic (AVL)* and the other is for *fixed voltage logic (FVL)*. An AVHDR example is shown in Fig. 5.1, in which a huddle, h, consists of AVL and FVL.

Adaptive Voltage Logic (AVL)

AVL is composed of several AVFUs, and the voltage of each AVFU can be changed.

Adaptive Voltage Functional Unit (AVFU): A dedicated functional unit and I/O level converters in h. Each pair of a functional unit and I/O level converters is connected to its dedicated power supply rail (Fig. 5.2). Its supply voltages are controlled by its PMOS header switches dynamically.

Fixed Voltage Logic (FVL)

FVL is composed of HLRs, FSM, and HLCs, and only one constant voltage is assigned to FVL.

Huddled Local Registers (HLRs): Dedicated local registers in h and input multiplexers. AVFUs can only access the HLRs in h. We ignore the interconnection dela lose to the HLRs.

Finite State Machine (FSM): A dedicated controller in h. FSM controls

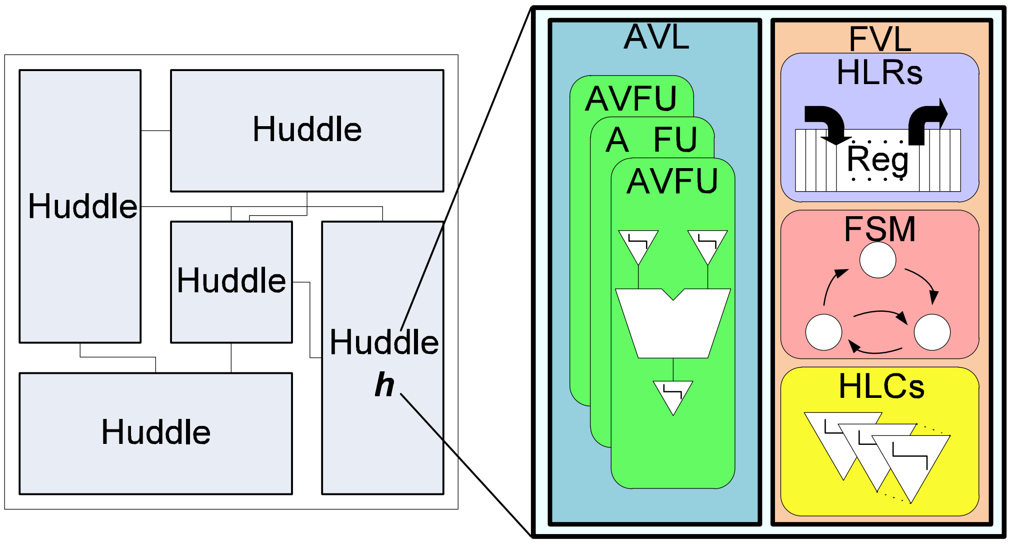


Figure 5.1: An AVHDR architecture.

the AVFUs and the HLRs in h. FSM control the supply voltages for the AVFUs at runtime.

Huddled Level Converters (HLCs): Dedicated level converters in h. HLCs are used during inter-huddle data transfer from lower voltage huddles.

The AVHDR architecture realizes two energy saving contributions:

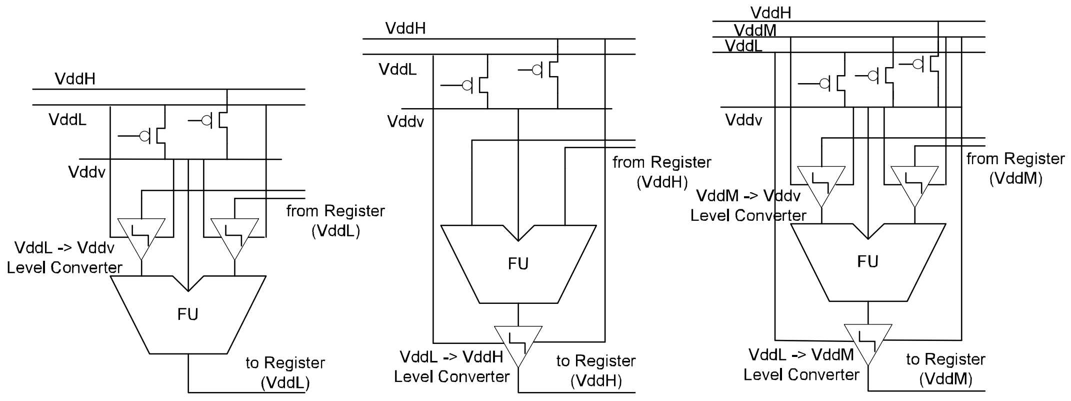
(1) Huddles realize the strategy which take interconnection delay into consideration in HLS with relatively little area overheads.

(2) AVL and FVL realize the smart supply voltages selection with dynamic multiple supply voltages.

Based on huddles, the number of non-critical operations can be increased by managing interconnection delay, and then optimized supply voltages can be assigned to AVL and FVL for energy reduction.

5.3 Problem Definition

To develop energy efficient HLS algorithms for the proposed AVHDR architecture, the problem could be modeled as follows. A control-data flow graph (CDFG) $G(N, E)$ is a directed graph, where a node set N is composed of an operation node set N_o and a branching control node set N_c (start and end nodes of conditional



(a) FU voltage can be higher than register voltage.

(b) FU voltage can be lower than register voltage.

(c) FU voltage can be both lower and higher than register voltage.

Figure 5.2: The number of AVFU level converters.

branches), and an edge set E is composed of a data-flow edge set E_d and a control-flow edge set E_c. Let $CV = \{00\ldots0 \le CV_b \le 11\ldots1\}$ be a set of a-bit condition vector (CV) [44], which is a bitwise encoding of distinct conditional branches. We consider the three supply voltages, v_l, v_m, and v_h ($v_l < v_m < v_h$), which are assigned to each operation node n_x. $V(n_x)$ is a supply voltage that is assigned to the operation node n_x. T_{clk} refers to a clock period constraint and S_{max} refers to a control step (CS) constraint. Let $S = \{1, \cdots, S_{max}\}$ be a set of CSs. Let S_t be a CS ($1 \le S_t \le S_{max}$).

Let $F = \{f_1, \cdots, f_p\}$ be a set of functional units. We consider the three supply voltages, v_l, v_m, and v_h ($v_l < v_m < v_h$), which are assigned to each functional unit in each step. Let v_{gate} be a state of power gating when none of the three voltages, v_l, v_m, and v_h are assigned to functional units. Let $D_f(f_i, v_l)$ be the delay of the functional unit f_i in F to which v_l is assigned. Let $E_d(f_i, v_l)$ be the dynamic energy consumed by the functional unit f_i to which v_l is assigned. Let $P_l(f_i, v_l)$ be the leak power consumed by the functional unit f_i to which v_l is assigned. Likewise, we can define $D_f(f_i, v_m)$, $E_d(f_i, v_h)$, $P_l(f_i, v_m)$, and so on. Let $SWE(f_i, v_l, v_h)$ be the energy consumed by the functional unit f_i when the supply voltage is switched from v_l to v_h. Likewise, we can define $SWE(f_i, v_m, v_h)$, $SWE(f_i, v_{gate}, v_l)$, and so on. Let $AVT(f_i, S_t, CV_b)$ be the supply voltage which is assigned to the functional unit f_i in the CS S_t of the conditional vector CV_b. Then the adaptive voltage table AVT can be defined by a $p \times S_{max} \times a$ matrix whose (i, t, b)-element is expressed by $AVT(f_i, S_t, CV_b)$.

Let $Gain(f_i, v_l, v_h)$ be the dynamic energy saving when the supply voltage of the functional unit f_i is changed from v_l to v_h. $Gain(f_i, v_l, v_h)$ is calculated by as follows:

$$Gain(f_i, v_l, v_h) = y\{E_d(f_i, v_h) - E_d(f_i, v_l)\} \tag{5.1}$$

where y is the number of executed operations when v_l is assigned to f_i.
Let $BET(f_i, v_l, v_h)$ be the break even time [36] when the supply voltage of the functional unit f_i is changed from v_l to v_h. $BET(f_i, v_l, v_h)$ is calculated by as follows:

$$BET(f_i, v_l, v_h) = \frac{SWE(f_i, v_l, v_h) - Gain(f_i, v_l, v_h)}{P_l(f_i, v_h) - P_l(f_i, v_l)}. \tag{5.2}$$

Let $H = \{h_1, \cdots, h_q\}$ be a set of huddles in the AVHDR architecture. Each functional unit is bound to any one of the huddles and the binding is defined by a function $Hud : F \rightarrow H$. $Hud(f_i)$ is the huddle to which f_i is bound. $F(h_j)$ is a set of functional units which are bound to h_j. $D_{reg}(h_j)$ is a delay of HLRs in h_j. We consider the three supply voltages, v_l, v_m, and v_h ($v_l < v_m < v_h$), which are assigned to each FVL. $V(h_j)$ is a supply voltage which is assigned to the FVL inside huddle h_j. $D_{lc}(v_l, v_m)$ is a delay of a level converter which changes the voltage from v_l to v_m. Likewise, we can define $D_{lc}(v_l, v_h)$, $D_{lc}(v_m, v_h)$, and so on.

$D_{total}(h_j, f_i, v_l)$ shows the delays required to execute the functional unit $f_i \in F(h_j)$ to which v_l is assigned. $D_{total}(h_j, f_i, v_l)$ is defined by:

$$D_{total}(h_j, f_i, v_l) = D_f(f_i, v_l) + D_{reg}(h_j) + D_{lc}(v_l, V(h_j)). \tag{5.3}$$

$S_f(h_j, f_i, v_l)$ shows the number of CSs required to execute the functional unit $f_i \in F(h_j)$ to which v_l is assigned. $S_f(h_j, f_i, v_l)$ is defined by:

$$S_f(h_j, f_i, v_l) = \left\lceil \frac{D_{total}(h_j, f_i, v_l)}{T_{clk}} \right\rceil. \tag{5.4}$$

$Slack(f_i, S_t, CV_b)$ is defined by:

$$\begin{aligned} Slack(f_i, S_t, CV_b) =& T_{clk} \cdot S_f(h_j, f_i, AVT(f_i, S_t, CV_b)) \\ & - D_f(f_i, AVT(f_i, S_t, CV_b)). \end{aligned} \tag{5.5}$$

$Slack(f_i, S_t, CV_b)$ shows the slack time of the functional unit $f_i \in F(h_j)$ which can be used by data transfer for succeeding operations in the CS S_t of the conditional vector CV_b. $MinSlack(f_j)$ is defined by:

$$MinSlack(f_i) = \min_{S_t \in S, CV_b \in CV} Slack(f_i, S_t, CV_b). \tag{5.6}$$

$MinSlack(f_j)$ shows the minimum slack time through all steps. The width and height of each huddle must satisfy the following *huddle size constraint*:

$$2 \cdot D_w(W(h_j) + H(h_j)) \leq \min_{f_i \in F(h_j)} \{MinSlack(f_i)\} \tag{5.7}$$

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 whose length is x. In the proposed algorithm, we obtain the value of $(W(h_j) + H(h_j))$ so that it satisfies the huddle size constraint and determine $W(h_j)$ and $H(h_j)$ by using the aspect ratio predefined for each huddle.

Let $Dist(h_j, h_k)$ be the Manhattan distance between the the center of huddles h_j and h_k. Then $D_w(Dist(h_j, h_k))$ shows the interconnection delay between them. 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 HLRs in h_k which is defined by:

$$Tr(f_i, h_k) = D_w(Dist(h_j, h_k)) + D_{lc}(V(h_j), V(h_k)) + D_{reg}(h_k). \tag{5.8}$$

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

$$DT(f_i, h_k) = \begin{cases} 0 \quad (MinSlack(f_i) \geq Tr(f_i, h_k)), \\ \lceil Tr(f_i, h_k)/T_{clk} \rceil \\ \qquad (MinSlack(f_i) < Tr(f_i, h_k)). \end{cases} \tag{5.9}$$

We prepare two types of data transfer mode:

Mode 1: In the case of $MinSlack(f_i) \geq Tr(f_i, h_k)$, the functional unit $f_i \in F(h_j)$ directly stores its output into the registers in the huddle h_k. Thus, the data transfer requires no extra cycles.

Mode 2: In the case of $MinSlack(f_i) < Tr(f_i, h_k)$, the functional unit $f_i \in F(h_j)$ first stores its output into the registers in the huddle h_j. In the subsequent cycles, we perform the data transfer from the huddle h_j to the huddle h_k. The data transfer requires $\lceil Tr(f_i, h_k)/T_{clk} \rceil$ cycles.

Then the data transfer table DT can be defined by a $p \times q$ matrix whose (i, k)-element is expressed by $DT(f_i, h_k)$.

Based on the above definitions, the HLS problem is defined as follows:

Definition 5.1. *The HLS problem is, for a given CDFG, a clock cycle constraint, a CS constraint, and a set of functional units, to assign each operation node to a CS and a functional unit to bind each functional unit to a huddle, and to assign a supply voltage to each operation, each functional unit, and each FVL, so that the given CDFG is executed correctly considering multi-cycle interconnect communications. The objective is to minimize the total energy consumption.* □

With the above definition, we can manipulate pipelined functional units. For instance, a two-stage multiplier can be managed by partitioned into three component as follows:

- The first half of the multiplier,
- the pipeline register,
- the last half of the multiplier.

The first half and the last half of the multiplier are considered to be AV-FUs, and the pipeline register is considered to be a HLR. Furthermore, pipelined multiplication-node n_m is split into the first half multiplication-node n_{m1} and the last half multiplication-node n_{m2}. A data-flow edge e_{mp} is inserted between n_{m1} and n_{m2}. According to the above transformation, the definition can be applied to pipelined functional units.

5.4 The SAAV Algorithm

In this section, a new high-level Synthesis Algorithm for Adaptive Voltage huddle-based distributed-register architecture called SAAV is proposed.

Generally, high-level synthesis algorithms considering multi-cycle interconnect communication are composed of schedulings, bindings, and floorplannings and classified into the following two types:

Type 1: Schedulings, bindings, and floorplannings are executed a predetermined number of times in a predetermined order.

Type 2: Schedulings, bindings, and floorplannings are executed repeatedly as an iterative refinement flow.

In Type 1, a required time to synthesize a chip can be expected easily since the number of executed synthesis steps and the execution order are determined. If we know in advance how many times we need to perform each high-level synthesis step as well as its best execution order, Type 1 will be the best choice. MCAS [8], one of the RDR architecture synthesis algorithms, uses an approach based on Type 1 above. Since RDR architecture contains uniform-sized islands, inter-island delays are unchanged even if RDR island configurations are changed. Then we can execute a predetermined design flow.

In Type 2, several informations such as scheduling results and placement results are fed back to each other since each synthesis step is executed repeatedly as many times as needed. A GDR architecture synthesis algorithm [32] and HDR architecture synthesis algorithms (Chapter 3 and Chapter 4) use approaches based on Type 2. By iteratively executing scheduling/binding steps and floorplanning steps, a current scheduling/binding step can consider interconnection delay obtained in a previous floorplanning step. The shape and size of each module are determined in a scheduling/binding step and a floorplanning step is done using these module informations. Because each synthesis step affects each other, iterative refinement flows as in Type 2 must be the best choice targeting GDR and HDR.

In SAAV, we must consider the design influence of level converters and power switches which are required for dynamic multiple supply voltages. It is very difficult to predict the influence of these components in algorithms based on Type 1. Therefore Type 2-based synthesis flow is suitable for the proposed AVHDR architecture, in which global optimization could be achieved by considering dynamic multiple supply voltages.

Based on Type 2, a virtual-area-based iterative refinement flow proposed in Chapter 4 is used in SAAV. SAAV is composed of the following seven steps in each iteration:

- **initial huddling,**
- **scheduling/binding,**
- **register/controller synthesis,**
- **huddle voltage adaptation,**
- **floorplanning,**
- **floorplanning-directed huddling,**
- **virtual area adaptation.**

In a initial huddling step, initial huddle configuration and placement are determined by given functional units. In a scheduling/binding step, each operation node in a CDFG is assigned to a CS and a functional unit considering dynamic multiple supply voltages and multi-cycle interconnect communications. In a register/controller synthesis step, HLR and FSM configurations in each huddle are determined using the scheduling/binding result. In huddle voltage adaptation step, we assign an optimal voltage to each FVL considering energy consumption of level converters. In a floorplanning step, every huddle is placed. In a floorplanning-directed huddling step, the configuration and placement of each huddle is determined simultaneously. In a virtual area adaptation step, we reduce the virtual area overhead during iteration. Fig. 5.3 shows the SAAV algorithm. In the rest of this section, we will describe the virtual-area-base iterative refinement flow and each step in Fig. 6.3. Note that we deal with only DFG, only two voltages such as v_h and v_l, and only one operation type for simplicity as a motivated example but we can deal with CDFG, v_m, and any other operation type similarly.

SAAV uses a virtual-area-based iterative refinement flow. Initially we prepare two types of area, called a real area and a virtual area. Let $A_{real}(h_j)$ be the real area of the huddle h_j, and which is the sum of functional unit areas and register areas inside h_j. Let $A_{virtual}(h_j)$ be the virtual area of huddle h_j, and it is estimated in the iteration when its huddle construction is changed as follows:

1. In each iteration, $A_{real}(h_j)$ is calculated by summing up the areas of functional units, registers, a controller, and level converters inside h_j.
2. If $A_{virtual}(h_j) \geq A_{real}(h_j)$, $A_{virtual}(h_j)$ is not updated.
3. If $A_{virtual}(h_j) < A_{real}(h_j)$, we set $A_{virtual}(h_j) = A_{real}(h_j)$.

At the beginning of the iteration process, $A_{virtual}(h_j)$ of the huddle h_j is initialized to be $A_{real}(h_j)$. SAAV is mainly composed of the three processes: initial process, iteration process, and adjustment process. In the initial process and the adjustment process, huddle placement is determined based on a real area of each huddle. In the iteration process, we perform scheduling/binding and floorplanning repeatedly based on virtual area of each huddle. When no timing violation occurs in the iteration process, we go to the adjustment process for real area estimation.

5.4.1 Initial huddling

In initial huddling, initial huddle configuration and placement are determined by given functional units (as shown in Fig. 5.4). If p functional units are given which we prepare just p huddles in which each functional unit is assigned to each huddle and the supply voltage v_h is assigned to each of AVLs and FVLs. All the huddles are overlapped with each other so as to ignore interconnection delay between huddles.

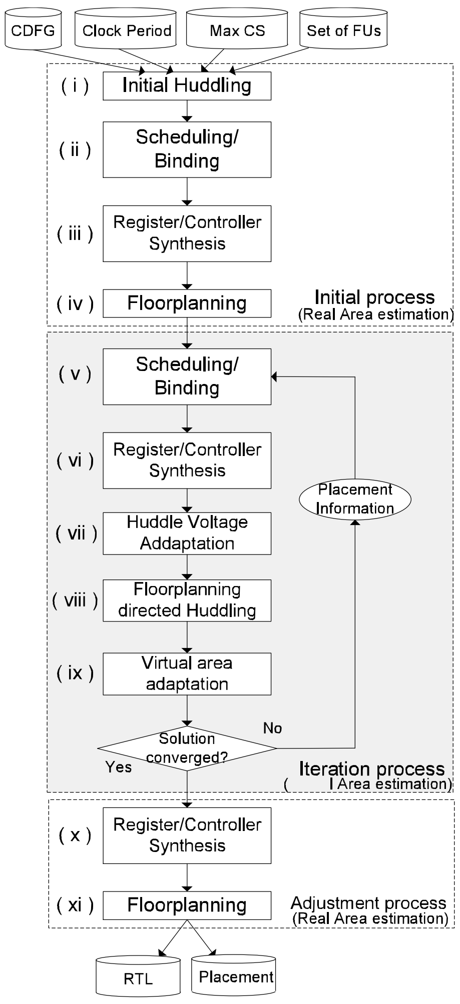


Figure 5.3: The SAAV algorithm.

Example 5.1. *In initial huddling, initial huddle configuration and placement are determine by a given set of functional units F. The input and the output are as follows:*

Input *:*

- $F = \{f_1, f_2, f_3, f_4\}$.

Output *:*

- *huddle configuration (Fig. 5.4).*

Because all the huddles are overlapped with each other, we can ignore the interconnection delay between huddles in the initial process.

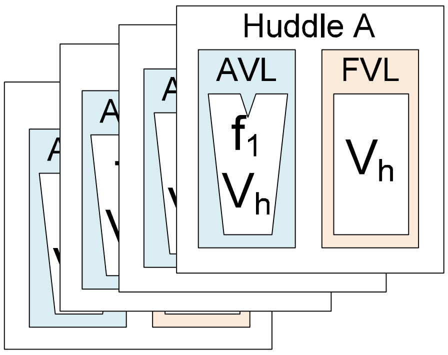


Figure 5.4: The result of initial huddling.

5.4.2 Scheduling/binding

The scheduling/binding problem is, for given a CDFG $G(N, E)$, a clock period constraint T_{clk}, a CS constraint S_{max}, a set of functional units, and huddle configuration, to find scheduling, functional unit binding, and supply voltage binding of every node in a given CDFG and to determine supply voltages assigned to given AVL and FVL so as to minimize the energy consumption of AVL and minimize the supply voltage of FVL.

The scheduling/binding is composed of three phases: (a) FVL voltage decreasing phase (Algorithm 5.1), (b) AVL voltage decreasing phase (Algorithm 5.2), and (c) dynamic voltage assignment phase (Algorithm 5.3). In all the phases, we use data-transfer-table-based list scheduling [31] in order to search for the optimal voltage. We can consider interconnection delay and conditional branch by using the scheduling.

In the FVL voltage decreasing phase, we design a priority $P_s(h_j)$ for a huddle h_j. The priority $P_s(h_j)$ is expressed as the sum of energy consumptions of AVFUs inside the huddle h_j and calculated by:

$$P_s(h_j) = \sum_{f_i \in F(h_j)} E(f_i) \tag{5.10}$$

We pick up the huddle whose priority $P_s(h_j)$ is the largest first and try to decrease its FVL voltage. We repeat this process unless a scheduled CDFG violates the CS constraint S_{max}.

Example 5.2. *This example shows that the step (v) just after the initial process.*

In scheduling/binding, operation scheduling and functional unit binding are executed. The inputs and the outputs are as follows:

Inputs *:*

Algorithm 5.1 (a) FVL voltage decreasing phase

```
1: Assign $V(h_j) \leftarrow v_h$ for each huddle $h_j$.
2: Execute scheduling/binding based on $DT(f_i, h_k)$ [31].
3: Calculate $P_s(h_j)$.
4: for $h_j$ in the descending-order of $P_s(h_j)$ do
5:   if $V(h_j) = v_h$ then
6:     $V(h_j) \leftarrow v_m$.
7:     Execute scheduling/binding based on $DT(f_i, h_k)$.
8:     if the resultant scheduling exceeds $S_{max}$ then
9:       $V(h_j) \leftarrow v_h$.
10:    end if
11:  end if
12: end for
13: for $h_j$ in the descending-order of $P_s(h_j)$ do
14:   if $V(h_j) = v_m$ then perform (4)–(12) with $V(h_j) \leftarrow v_l$.
15: end for
```

Table 5.1: Component information.

	v_h	v_l
FU	1.0 ns	2.0 ns
REG	0.2 ns	0.4 ns

	$v_l \rightarrow v_h$
LC	0.1 ns

- *DFG $G(N, E)$ (Fig. 5.5(a)),*
- *Component information (Table 5.1),*
- $T_{clk} = 1.5$ ns,
- $S_{max} = 4$, *and*
- *huddle configuration (Fig. 5.5(b)).*

Outputs *:*

- *scheduled DFG (Fig. 5.9(a)),*
- *adaptive voltage table AVT (Fig. 5.9(b)), and*
- *voltage assignment to FVL (Fig. 5.9(c)).*

First, we assume interconnection delay between huddles by the huddle placement of step (iv). According to the example showing in Fig 5.6(c), the interconnection

Algorithm 5.2 (b) AVL voltage decreasing phase

```
1: Assign $AVT(f_i, S_t, 11\ldots1) \leftarrow v_h$ for each AVFU $f_i$ and each CS $S_t$.
2: for $f_i$ in the descending-order of $E_d(f_i)$ do
3:   if $AVT(f_i, 1, 11\ldots1) = v_h$ then
4:     for $S_t$ in $S$ do
5:       $AVT(f_i, S_t, 11\ldots1) \leftarrow v_m$.
6:     end for
7:     Execute scheduling/binding based on $DT(f_i, h_k)$.
8:     if the resultant scheduling exceeds $S_{max}$ then
9:       for $S_t$ in $S$ do
10:        $AVT(f_i, S_t, 11\ldots1) \leftarrow v_h$.
11:      end for
12:    end if
13:  end if
14: end for
15: for $f_i$ in the descending-order of $E_d(f_i)$ do
16:   if $AVT(f_i, 1, 11\ldots1) = v_m$ then perform (2)–(14) with
      $AVT(f_i, S_t, 11\ldots1) \leftarrow v_l$.
17: end for
```

delay can be assumed as follows:

$$\begin{aligned} &D_w(Dist(A,B)) = D_w(Dist(A,C)) \\ &= D_w(Dist(B,D)) = D_w(Dist(C,D)) = 0.2\,\text{ns}, \end{aligned} \tag{5.11}$$

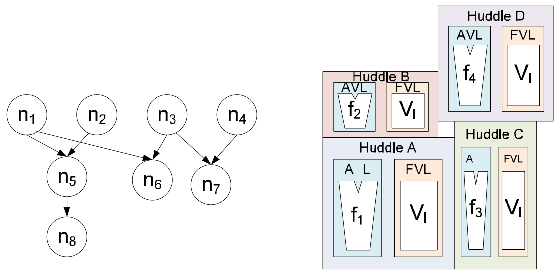
$$D_w(Dist(A,D)) = D_w(Dist(B,C)) = 0.4\,\text{ns}. \tag{5.12}$$

FVL voltage decreasing phase *Algorithm 5.1 shows the FVL voltage decreasing phase algorithm. First, the voltage v_h is assigned to each functional unit and each register as shown in Fig. 5.6(c). Next, $DT(f_i, h_k)$ is calculated. $D_{total}(A, f_1, v_h)$ can be calculated by:*

$$\begin{aligned} D_{total}(A, f_1, v_h) =& D_f(f_1, v_h) + D_{reg}(A) + D_{lc}(v_h, V(A)) \\ =& 1.0\,\text{ns} + 0.2\,\text{ns} + 0\,\text{ns} \\ =& 1.2\,\text{ns}. \end{aligned}$$

Algorithm 5.3 (c) Dynamic voltage assignment phase

```
1: Execute scheduling/binding based on $DT(f_i, h_k)$.
2: for $N_x$ in $N_o$ do
3:   if $V(N_x) = v_m$ then
4:     Let $FLAG \leftarrow false$.
5:     for $f_i$ in $F$ do
6:       for $S_t$ in $S$ do
7:         for $CV_b$ in $CV$ do
8:           if $AVT(f_i, S_t, CV_b) = v_l$ then
9:             $FLAG \leftarrow true$.
10:          end if
11:        end for
12:      end for
13:    end for
14:    if $FLAG = true$ then
15:      Execute scheduling/binding based on $DT(f_i, h_k)$ without changing any
         other operation voltages.
16:      if the resultant scheduling does not exceeds $S_{max}$ then
17:        go to next $N_x$.
18:      end if
19:    end if
20:    $V(N_x) \leftarrow v_l$ not changing the binding result of (1) utilizing dynamic
       multiple supply voltages.
21:    Execute scheduling/binding based on $DT(f_i, h_k)$ without changing any
       other operation voltages.
22:    if the resultant scheduling exceeds $S_{max}$ then
23:      $V(N_x) \leftarrow v_h$.
24:    end if
25:  end if
26: end for
27: for $N_x$ in $N_o$ do
28:   if $V(N_x) = v_h$ then perform (2)–(26) with $V(N_x) \leftarrow v_l$.
29: end for
30: for $N_x$ in $N_o$ do
31:   if $V(N_x) = v_h$ then perform (2)–(26) with $V(N_x) \leftarrow v_m$.
32: end for
33: Reassign $AVT(f_i, S_t, CV_b)$ with the scheduling/binding result in this phase.
```



(a) Input DFG. (b) The huddle configuration.

Figure 5.5: The inputs of scheduling/binding.

$S_f(A, f_1, v_h)$ is calculated by:

$$\begin{aligned} S_f(A, f_1, v_h) &= \left\lceil \frac{D_{total}(A, f_1, v_h)}{T_{clk}} \right\rceil \\ &= \left\lceil \frac{1.2\,\text{ns}}{1.5\,\text{ns}} \right\rceil \\ &= 1. \end{aligned}$$

Because there are no conditional branches and dynamically voltage changing, $MinSlack(f_1)$ is calculated by:

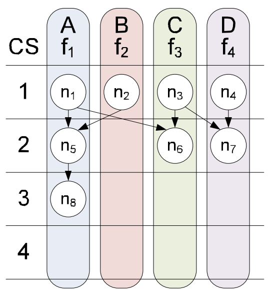
$$\begin{aligned} MinSlack(f_1) &= T_{clk} \cdot S_f(A, f_1, v_h) - D_f(f_1, v_h) \\ &= 1.5\,\text{ns} \cdot 1 - 1.0\,\text{ns} \\ &= 0.5\,\text{ns}. \end{aligned}$$

$Tr(f_1, B)$ is calculated by:

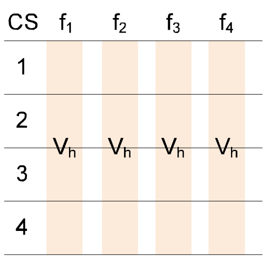
$$\begin{aligned} Tr(f_1, B) &= D_w(Dist(A, B)) + D_{lc}(V(A), V(B)) + D_{reg}(B) \\ &= 0.2\,\text{ns} + 0\,\text{ns} + 0.2\,\text{ns} \\ &= 0.4\,\text{ns}. \end{aligned}$$

Because $MinSlack(f_1) \geq Tr(f_1, B)$, the data transfer Mode 1 is selected and the data transfer time is calculated to be $DT(f_1, B) = 0$. Similarly, Mode 1 is applied to the data transfer from f_1 to C and $DT(f_1, C) = 0$. On the other hand, $Tr(f_1, D)$ is calculated by:

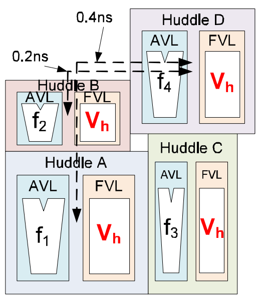
$$\begin{aligned} Tr(f_1, D) &= D_w(Dist(A, D)) + D_{lc}(V(A), V(D)) + D_{reg}(D) \\ &= 0.4\,\text{ns} + 0\,\text{ns} + 0.2\,\text{ns} \\ &= 0.6\,\text{ns}. \end{aligned}$$



(a) The scheduled DFG.



(b) The adaptive voltage table.



(c) The huddle configuration.

	A	B	C	D
f_1	0	0	0	1
f_2	0	0	1	0
f_3	0	1	0	0
f_4	1	0	0	0

(d) The data transfer table.

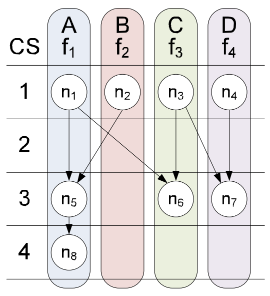
Figure 5.6: The result of Algorithm 5.1(1–2).

Because $MinSlack(f_1) < Tr(f_1, D)$, the data transfer Mode 2 is selected and the data transfer time $DT(f_1, D)$ is calculated by:

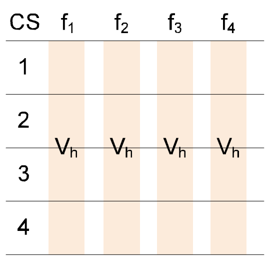
$$\begin{aligned} DT(f_1, D) =& \lceil Tr(f_i, h_k)/T_{clk} \rceil \\ =& \lceil 0.6\,\text{ns}/1.5\,\text{ns} \rceil \\ =& 1. \end{aligned}$$

In this way, $DT(f_i, h_k)$ is constructed as shown in Fig. 5.6(d). Based on the $DT(f_i, h_k)$, the input DFG is scheduled as shown in Fig. 5.6(a).

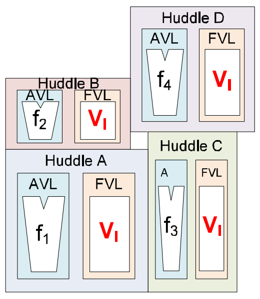
Next, we pick up the huddle D in Fig. 5.6(c) with the voltage v_h and change V(D) from v_h to v_l. Since the scheduling result satisfies the CS constraint S_{max}, we pick up the huddle C, huddle B, and huddle A and change their voltages similarly. Finally, Fig. 5.7(c) shows the huddle configuration, Fig. 5.7(d) shows the constructed data transfer table $DT(f_i, h_k)$, and the DFG is scheduled as shown in Fig. 5.7(a). Since there are no huddles with the voltage v_h, we execute the AVL voltage decreasing phase next. Note that in this phase, the voltage v_h is assigned to each functional unit in each CS(Figs. 5.6(b) and 5.7(b)).



(a) The scheduled DFG.



(b) The adaptive voltage table.



(c) The huddle configuration.

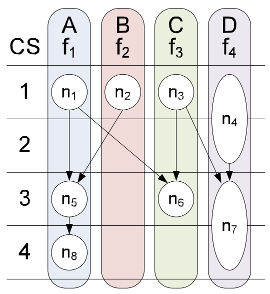
	A	B	C	D
f_1	0	1	1	1
f_2	1	0	1	1
f_3	1	1	0	1
f_4	1	1	1	0

(d) The data transfer table.

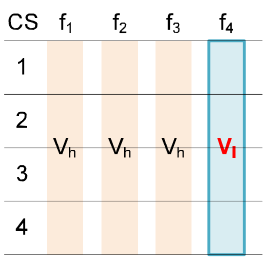
Figure 5.7: The result of FVL voltage decreasing phase.

AVL voltage decreasing phase *Algorithm 5.2 shows the AVL voltage decreasing phase algorithm. First, we pick up the functional unit f_4 of huddle D in Fig. 5.7(c) with the voltage v_h and change f_4 voltage from v_h to v_l (Fig. 5.8(b)). Fig. 5.8(d) shows the constructed data transfer table and the DFG is scheduled as shown in Fig. 5.8(a). The result satisfies the CS constraint S_{max}. Next, we pick up the functional unit f_3 of huddle C in Fig. 5.7(c) with the voltage v_h and change f_3 voltage from v_h to v_l but the scheduling result exceeds the CS constraint S_{max}. Therefore we return the voltage of f_3 from v_l to v_h. We pick up the functional unit f_2 and f_1 and change their voltage similarly, but all the results cannot satisfy the CS constraint S_{max}. Finally, we get the scheduled DFG as shown in Fig. 5.8(a), Fig. 5.8(b) shows the adaptive voltage table, Fig. 5.8(c) shows the huddle configuration, and Fig. 5.8(d) shows the data transfer table. We execute the dynamic voltage assignment phase next.*

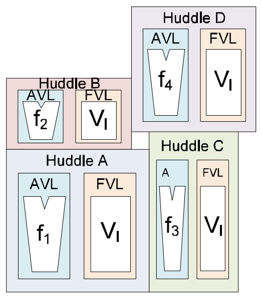
Dynamic voltage assignment phase *Algorithm 5.3 shows the dynamic voltage assignment phase algorithm. First, we pick up the node n_1 in Fig. 5.8(a) with the voltage v_h and change n_1 voltage from v_h to v_l. We execute scheduling/binding based on $DT(f_i, h_k)$ but the result exceeds the CS constraint S_{max}. Next, we assign*



(a) The scheduled DFG.


CS
f_1
f_2
f_3
f_4
1
2
3
4
V_h
V_h
V_h
V_l

(b) The adaptive voltage table.



(c) The huddle configuration.

	A	B	C	D
f_1	0	1	1	1
f_2	1	0	1	1
f_3	1	1	0	1
f_4	1	1	1	0

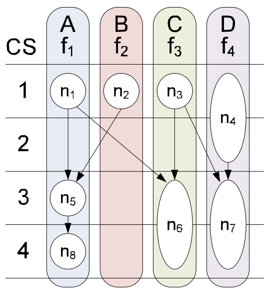
(d) The data transfer table.

Figure 5.8: The result of AVL voltage decreasing phase.

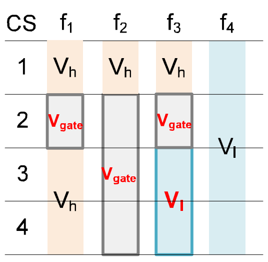
the voltage v_l to n_1 not changing the binding result of Fig. 5.8(a). We execute scheduling/binding based on $DT(f_i, h_k)$ again but the result exceeds the CS constraint S_{max}. We pick up the node $n_2, \cdots, n_5$ and change their voltage similarly but the scheduling result exceeds the CS constraint.

When we pick up the node n_6, we can assign the voltage v_l to n_6 with dynamic multiple supply voltages. First, we pick up the node n_6 in Fig. 5.8(a) and change n_6 voltage from v_h to v_l. We execute scheduling/binding based on $DT(f_i, h_k)$ but the result exceeds the CS constraint S_{max}. Next, we assign the voltage v_l to n_6 not changing the binding result of Fig. 5.8(a). We execute scheduling/binding based on $DT(f_i, h_k)$ again then the result can satisfy the CS constraint S_{max}(Fig. 5.9(a)).

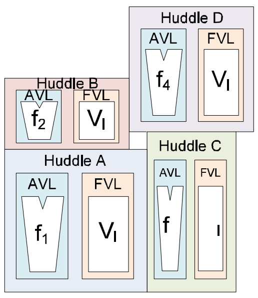
We repeat this step on the node n_7 and n_8 but the scheduling result exceeds the CS constraint. Finally, we get the scheduled DFG as shown in Fig. 5.9(a), Fig. 5.9(c) shows the huddle configuration, and Fig. 5.9(d) shows the data transfer table. Based on the scheduled DFG (Fig. 5.9(a)), we assign the adaptive voltage table AVT as shown in Fig. 5.9(b).



(a) The scheduled DFG.



(b) The adaptive voltage table.



(c) The huddle configuration.

	A	B	C	D
f_1	0	1	1	1
f_2	1	0	1	1
f_3	1	1	0	1
f_4	1	1	1	0

(d) The data transfer table.

Figure 5.9: The result of dynamic voltage assignment phase. Fig. 5.9 (a) shows the final result of scheduling/binding. Figs. 5.9 (b)–(d) show its associated adaptive voltage table, huddle configuration, and data transfer table.

5.4.3 Register/controller synthesis

In the register/controller synthesis step, the register and controller configuration in each huddle is determined according to the result of the previous scheduling/binding step.

In controller synthesis, first we synthesize the adaptive voltage table AVT by taking BET of each pair of voltages into consideration. Algorithm 5.4 shows the algorithm of BET-aware AVT synthesis. Based on the synthesized AVT, the signals for power switching and control signals for multiplexers are synthesized.

Example 5.3. *In register/controller synthesis, the register and controller configuration in each huddle is determined according to the result of scheduling/binding step. The inputs and the outputs are as follows:*

Input *:*

- *BET information (Table 5.2),*
- *scheduled DFG (Fig. 5.9(a)), and*

Algorithm 5.4 BET-aware AVT synthesis

```
1: for functional unit $f_i \in F$ do
2:     if $f_i$ executes operation less than $BET(f_i, v_l, v_m)$ in the CS $S_t$ and the conditional vector $CV_b$ then
3:         $AVT(f_i, S_t, CV_b) \leftarrow v_m$.
4:     end if
5:     if $f_i$ executes operation less than $BET(f_i, v_l, v_h)$ in the CS $S_t$ and the conditional vector $CV_b$ then
6:         $AVT(f_i, S_t, CV_b) \leftarrow v_h$.
7:     end if
8:     if $f_i$ executes operation less than $BET(f_i, v_m, v_h)$ in the CS $S_t$ and the conditional vector $CV_b$ then
9:         $AVT(f_i, S_t, CV_b) \leftarrow v_h$.
10:    end if
11:    if $f_i$ sleep longer than $BET(f_i, v_{gate}, v_{after})$ in the CS $S_t$ and the conditional vector $CV_b$ where $v_{afeter}$ is the assign voltage after sleep then
12:        Do nothing.
13:    else if $f_i$ sleep longer than $BET(f_i, v_l, v_{after})$ in the CS $S_t$ and the conditional vector $CV_b$ where $v_{afeter}$ is the assign voltage after sleep then
14:        $AVT(f_i, S_t, CV_b) \leftarrow v_l$.
15:    else if $f_i$ sleep longer than $BET(f_i, v_m, v_{after})$ in the CS $S_t$ and the conditional vector $CV_b$ where $v_{afeter}$ is the assign voltage after sleep then
16:        $AVT(f_i, S_t, CV_b) \leftarrow v_m$.
17:    else
18:        $AVT(f_i, S_t, CV_b) \leftarrow v_h$.
19:    end if
20: end for
```

Table 5.2: BET informations

	$v_{gate} \to v_l$	$v_{gate} \to v_h$	$v_l \to v_h$
BET	2.5 ns(= 2 steps)	3.5 ns(= 3 steps)	2.0 ns(= 2 steps)

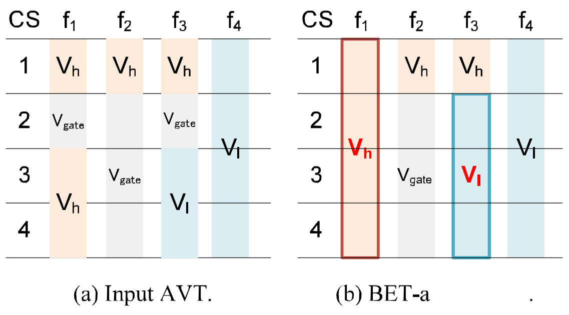


Figure 5.10: The result of BET-aware AVT synthesis.

- *adaptive voltage table AVT (Fig. 5.9(b)).*

Output *:*

- *huddle configuration (Fig. 5.11) and*
- *adaptive voltage table AVT (Fig. 5.10(b)).*

Algorithm 5.4 shows the BET-aware AVT synthesis algorithm. First, we pick up functional unit f_1. The sleep step of f_1 is CS 2. Because the sleep step is shorter than $BET(f_1, v_{gate}, v_h)$ and $BET(f_1, v_l, v_h)$, v_h is assigned to $AVT(f_1, 2, 1)$ (because there are no conditional branches, $CV_b = 1$). Next, we pick up functional unit f_2. The sleep steps of f_2 are CS 2–CS 4. Because the sleep steps are longer than $BET(f_2, v_{gate}, v_h)$, v_{gate} is assigned to $AVT(f_2, 2, 1)$, $AVT(f_2, 3, 1)$, and $AVT(f_2, 4, 1)$. Then we pick up functional unit f_3. The sleep step of f_3 is CS 2. The sleep step is shorter than $BET(f_3, v_{gate}, v_l)$, but is longer than $BET(f_3, v_l, v_l)(= 0)$. Therefore v_l is assigned to $AVT(f_3, 2, 1)$. Finally, we pick up functional unit f_4 but it has no sleep steps. The controller configuration in each huddle is determined by the synthesized AVT.

In the iteration process, we estimate each huddle area based on virtual area. In Fig. 5.11, the area of huddle A in this phase is smaller than that in the initial process. However, the virtual area of huddle A does not decrease in Fig. 5.11.

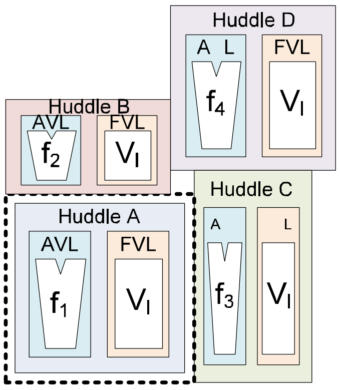


Figure 5.11: The result of register/controller synthesis.

5.4.4 Huddle voltage adaptation

In the huddle voltage adaptation step, the supply voltage of each FVL is reassigned so as to minimize the total energy consumption. In the scheduling/binding step, we minimize each FVL voltage. However, level converters of AVFU and HLC may consume wasted energy when we just assign the minimum voltage to each FVL. We should optimize the supply voltage of each FVL in order to minimize the total energy consumption including level converter energy. Since each FVL voltage is minimized in scheduling/binding step, we just consider increasing the FVL voltage accordingly. We pick up the voltage increasing FVL by using $P_s(h_j)$ which is the same value in the scheduling/binding step and try to increase the FVL voltage. If the total energy consumption decreases, we really assign the increasing voltage to the FVL. We repeat this process until all the huddles' FVL voltages have been examined as shown in Fig. 5.12.

Example 5.4. *In huddle voltage adaptation step, we reassign the supply voltage of FVL. The inputs and the output are as follows:*

Inputs *:*

- *scheduled DFG (Fig. 5.9(a)) and*
- *huddle configuration (Fig. 5.11).*

Output *:*

- *huddle configuration (Fig. 5.12).*

In scheduling/binding, FVL voltage decreasing phase minimize the FVL voltage of each huddle. However, level converters of AVFU and HLC may consume wasted

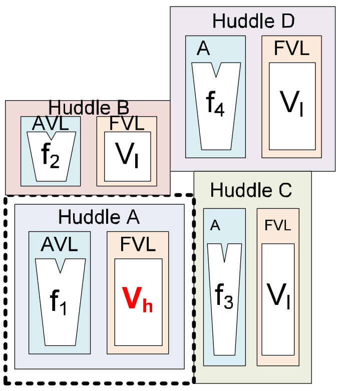


Figure 5.12: The result of huddle voltage adaptation.

energy when we just assign the minimum voltage of each FVL. In Fig. 5.9(a) and Fig. 5.9(c), the voltage v_l is assigned to the FVL of huddle A. In this case, we must use level converters of AVFU and HLC in n_1, n_5, n_8. If we assign the voltage v_h to huddle A, we do not use level converters and can decrease the total energy consumption. We change the FVL voltage of huddle A from v_l to v_h and get the huddle configuration as shown in Fig. 5.12.

5.4.5 Floorplanning and floorplanning-directed huddling

The huddle is the partition on the floorplanning which express the area where data transfers inside huddle finish within one clock cycle. It is better for us to integrate huddle construction methods into floorplanning. In floorplanning-directed huddling, huddle placement as well as its height and width are optimized by using a simulated annealing (SA) strategy based on a sequence-pair representation. In this step, we consider the four moves as follows:

Move 1: Select two elements and exchange them in Γ_+.

Move 2: Select two elements and exchange them in Γ_+ and Γ_-.

Move 3: Select one element and change its aspect ratio.

Move 4: Select functional unit f_i and transfer it from the huddle h_j to the huddle $h_k (\neq h_j)$.

In floorplanning, we only consider Move 1, Move 2, and Move 3. In SA optimization, its cost function *cost* is expressed by

$$cost = \frac{A_{BB}}{A_{total}} + \alpha \frac{V}{T_{clock}} + \beta \frac{W}{W_{MAX}} + \gamma \frac{A_{PNR}}{A_{total}} \tag{5.13}$$

where A_{BB} is the rectangle area which includes all the huddles (dead space may be included), A_{total} is the sum of huddles' area (dead space is not included), T_{clock} is the clock period constraint, V is the sum of violations of clock period constraint, W is the wire length, W_{MAX} is the max wire length calculated by (rectangle area's height + width)$\times$the number of wires, and A_{PNR} is the sum of power network resource. α, β and γ are parameters.

The initial solution of floorplanning and floorplanning-directed huddling at each iteration is the solution represented by its sequence-pair of the previous result so that the entire iteration in Fig. 5.3 can converge gradually. Initial temperature T_i in floorplanning and floorplanning-directed huddling at the i-th iteration of the synthesis flow is computed by

$$T_{i+1} = KT_i \tag{5.14}$$

where K is also a parameter and set to be $K < 1$.[1]

Example 5.5. *In floorplanning directed huddling, huddle placement and huddle configuration are optimized simultaneously. The input and the output are as follows:*

Input *:*

- *huddle configuration (Fig. 5.12).*

Output *:*

- *huddle configuration (Fig. 5.13).*

In the example, functional unit f_4 transfers into huddle C.

5.4.6 Virtual area adaptation

Virtual area may increase interconnection delay between huddles as the iterations proceed. To solve this problem, we should gradually decrease the difference between virtual area and real area.

[1]In the experiments, $\alpha = 100$, $\beta = 1$, $\gamma = 0.5$ and $K = 0.9$ were set. In Eqn. (5.13) $\alpha = 100$ was set because the most important objective is to eliminate timing violations. Wire length is correlated with wire dynamic energy consumption, and $\frac{A_{BB}}{A_{total}}$ and $\frac{W}{W_{MAX}}$ are affected by the wire length. Therefore β was set to be 1 so as to evaluate them equally. Since $\frac{A_{PNR}}{A_{total}}$ contributes to merging huddles (Move 4), some weights were putted on it and $\gamma = 0.5$ was set. Some conventional methods [31,32] based on iterative refinement flow set $K = 0.9$ in their experimental evaluations. Therefore in Eqn. (5.14) $K = 0.9$ was also set in the experimental evaluation. These parameters may not be optimum but the experimental results show that the proposed algorithm obtains the best results compared with the conventional methods.

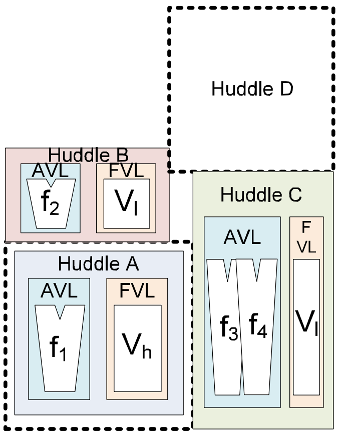


Figure 5.13: The result of floorplanning directed huddling.

We execute *virtual area adaptation* after floorplanning-directed huddling. Because this step is just before scheduling/binding at the next iteration, we can use virtual area closer to real area at the next iteration.

Virtual area adjustment is executed as follows:

1. Let $A_{dif}(h_j) = A_{virtual}(h_j) - A_{real}(h_j)$ be the difference between real area $A_{real}(h_j)$ and virtual area $A_{virtual}(h_j)$ of the huddle h_j.
2. We set $A_{virtual}(h_j) = A_{real}(h_j) + \phi \cdot A_{dif}(h_j)$.

where ϕ is an adaptive parameter. In order to decrease ϕ as the iterations proceed, we set $\phi = \max\{1 - 0.09i, 0\}$ at the i-th iteration.

Example 5.6. *In virtual area adaptation, we decrease the virtual area. The input and the output are as follows:*

Input *:*

- *huddle configuration (Fig. 5.13).*

Output *:*

- *huddle configuration (Fig. 5.14).*

In the example, the virtual area of huddle A and D are decreased.

In this example, since no timing violations occur, we go to the adjustment process.

5.5 Experimental Results

In this section, the circuit models are described and the proposed algorithm are evaluated.

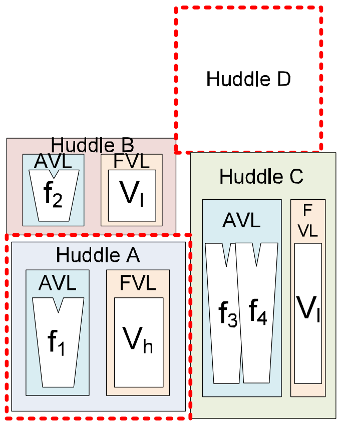


Figure 5.14: The result of virtual area adaptation.

The dynamic multiple supply voltages model

The circuit informations using the dynamic multiple supply voltages were obtained by Synopsys Design Compiler and Synopsys HSPICE based on CMOS 90 nm technology. First, functional units, such as Adder and Subtractor, were synthesized by Synopsys Design Compiler and the gate-level descriptions are obtained. Next, the gate-level descriptions were converted to the SPICE netlists. Then multiple power rails and PMOS header switches were inserted to the SPICE netlists and the circuits were simulated by Synopsys HSPICE. The values in Table 5.3 were obtained by the results of Synopsys Design Compiler and Synopsys HSPICE.

The level converter model

The circuit informations of level converters were obtained by Synopsys HSPICE based on CMOS 90 nm technology. The SPICE netlists based on a level converter circuit shown in Fig. 2.1 were described and the circuits were simulated by Synopsys HSPICE. The values in Table 5.5 were obtained by the results of Synopsys HSPICE. In Chapter 5 and Chapter 6, level converters are only inserted when data is transfered from lower voltages to higher voltages.

The results

The proposed algorithm have been implemented in C++ on UNIX 2.5 GHz $\times 2$ with 16 GB memory. The algorithm has been applied to DCT (a discrete cosine transform algorithm for 8×8 pixels, 48 nodes), EWF3 (three elliptic wave filters are serially connected, 102 nodes), FIR filter (a seventh order finite impulse response filter, 75 nodes), PARKER [29] (22 nodes, including conditional branches), and COPY (provided by a company, 378 nodes, including conditional branches). Comparing the experimental environment with the ones in [6–8, 17–19, 25, 26, 31, 32, 34, 35, 38, 47, 48], we cannot always say that the experimental environment is practical enough in terms of application size, but we can say that it is reasonable to demonstrate the effectiveness of the algorithm. Table 5.3 shows the functional units specification. Table 5.4 shows the registers and the multiplexers specification. Table 5.5 shows the level converters specification. Memories were assumed to be prepared outside. Therefore the memory access was assumed to be a special type of functional unit as shown in Table 5.3. All the functional units were assumed to have a bit width of 16. All the functional units, power switch [42], and BET were obtained based on CMOS 90 nm technology. Selectable voltages were assumed to be as $v_l = 0.8\,\mathrm{V}$, $v_m = 1.0\,\mathrm{V}$, and $v_h = 1.2\,\mathrm{V}$. The clock period constraint was given to be 1.5 ns in all experiments. The interconnection delay were assumed to be a proportion to square of the wiring length and an interconnection delay is set to be 1 ns when wiring length is $250\mu\mathrm{m}^2$ [32].

Table 5.6 and Table 5.7 summarize the experimental results. In Table 5.6, "S_{max}" shows the CS constraint S_{max} and "CPU time" shows CPU time to synthesize each circuit. "Average PG steps" in Table 5.7 represents the average power gating steps(= the sum of the power gating steps of all the functional units / the number of the functional units). In Table 5.7, "Dynamic energy", "Leak energy", and "Wire energy" represent dynamic energy consumption, leakage energy consumption, and wire dynamic energy consumption. "All energy" shows the sum of "Dynamic energy", "Leak energy", and "Wire energy". "Imp." shows the improvement of energy reduction over previous works.

The proposed SAAV algorithm (*SAAV* in Table 5.6 and Table 5.7) has been compared to a traditional shared-register architecture synthesis algorithm [31], GDR architecture synthesis algorithm [32], MCAS for RDR architectures [8], HDR architecture synthesis algorithm with single supply voltage (*MH*^{4} *(Single)* in Table 5.6 and Table 5.7), HDR architecture synthesis algorithm with multiple supply voltages (*MH*^{4} in Table 5.6 and Table 5.7). The proposed algorithm has been further compared with the following strategy: the existing dynamic-multiple-supply-

[2]Huddle size constraint (Eqn. (5.7)) of a huddle h_j is calculated by the clock period constraint $T_{clk} = 1.5\,\mathrm{ns}$, the interconnection delay, Table 5.3, Table 5.4, and Table 5.5. In general, huddle size will be minimized when the huddle contains the functional unit which gives the minimum slack defined by Eqn. (5.6). Huddle size will be maximized when the AVL in the huddle only contains the functional unit which gives the maximum slack. In this experiment, the minimum huddle size is $W(h_j) + H(h_j) = 41\,\mu\mathrm{m}$ (h_j contains 0.8 V-shifters and 0.8 V-FVL), and the maximum huddle size is $W(h_j) + H(h_j) = 152\,\mu\mathrm{m}$ (h_j only contains 1.2 V–16bit AND gates and 0.8 V–FVL).

Table 5.3: Informations of functional units.

Adder	Area :	386	μm^2
	1.2V	1.0V	0.8V
Delay[ns]	0.75	1.22	2.71
Dynamic energy[fJ]	103.97	64.00	33.22
Leak power[μW]	5.97	3.20	1.74
from 1.0V [fJ]	23.69	–	–
from 0.8V [fJ]	77.62	14.58	–
from 0V [fJ]	81.06	47.85	28.99
BET[ns]	13.58	14.95	16.65
Subtractor	Area :	417	μm^2
	1.2V	1.0V	0.8V
Delay[ns]	0.78	1.27	2.82
Dynamic energy[fJ]	109.49	67.40	34.99
Leak power[μW]	6.53	3.50	1.90
from 1.0V [fJ]	24.94	–	–
from 0.8V [fJ]	81.74	15.36	–
from 0V [fJ]	85.37	50.39	30.53
BET[ns]	13.08	14.40	16.03
Multiplier	Area :	2161	μm^2
	1.2V	1.0V	0.8V
Delay[ns]	1.65	2.70	6.00
Dynamic energy[fJ]	1324.38	788.00	495.13
Leak power[μW]	29.70	16.50	8.25
from 1.0V [fJ]	305.87	–	–
from 0.8V [fJ]	999.09	188.33	–
from 0V [fJ]	1136.25	728.75	373.51
BET[ns]	38.26	44.17	45.27
Shifter	Area :	294	μm^2
	1.2V	1.0V	0.8V
Delay[ns]	0.54	0.89	1.98
Dynamic energy[fJ]	84.64	52.10	27.05
Leak power[μW]	3.92	2.10	1.14
from 1.0V [fJ]	19.28	–	–
from 0.8V [fJ]	63.19	11.87	–
from 0V [fJ]	65.99	38.95	23.60
BET[ns]	16.85	18.55	20.66
Comparator	Area :	116	μm^2
	1.2V	1.0V	0.8V
Delay[ns]	0.51	0.83	1.84
Dynamic energy[fJ]	19.17	11.80	6.13
Leak power[μW]	1.25	0.67	0.36
from 1.0V [fJ]	4.37	–	–
from 0.8V [fJ]	14.31	2.69	–
from 0V [fJ]	14.95	8.82	5.35
BET[ns]	11.96	13.17	14.68
16bit AND	Area :	68	μm^2
	1.2V	1.0V	0.8V
Delay[ns]	0.4	0.66	1.47
Dynamic energy[fJ]	5.85	3.60	1.87
Leak power[μW]	1.23	0.66	0.36
from 1.0V [fJ]	1.33	–	–
from 0.8V [fJ]	4.37	0.82	–
from 0V [fJ]	4.56	2.69	1.63
BET[ns]	3.70	4.08	4.54
Memory access	Area :	–	μm^2
	1.2V	1.0V	0.8V
Delay[ns]	2.70	–	–

Table 5.4: Informations of registers. and multiplexers.

16bit Register	Area :	330	μm^2
	1.2V	1.0V	0.8V
Delay[ns]	0.29	0.47	1.05
Dynamic energy[fJ]	305.22	187.88	97.53
Leak power[μW]	2.75	1.47	0.80

16bit MUX	Area :	576	μm^2
	1.2V	1.0V	0.8V
Delay[ns]	0.13	0.21	0.47
Dynamic energy[fJ]	162.72	100.16	52.00
Leak power[μW]	16.65	8.93	4.86

Table 5.5: Informations of level converters.

V_{in}–V_{out}	Area [μm^2]	Delay [ns]	Dynamic energy [fJ]	Leak poewr [μW]
1.0V–1.2V	113	0.0423	1.92	2.473
0.8V–1.2V	113	0.0737	3.257	2.118
0.8V–1.0V	113	0.0589	2.221	3.553

voltage-a ltage assignment first was performed to each operation [6]; based on this voltage assignment, the proposed algorithm targeting for AVHDR architectures ([6] + AVHDR in Table 5.6 and Table 5.7) was performed. Because the algorithm in [6] only consider voltage assignment, the propose register/controller synthesis and floorplanning was performed after [6] for the energy comparison.

Table 5.9 and Table 5.8 show the huddle configurations of the experimental results. In Table 5.9 and Table 5.8, “FU” shows the functional units in each huddle. “AVL” shows the voltage assigned to functional units. “#REG” shows the number of registers in each huddle. “FVL” shows the voltage which is assigned to the FVL. MH^4 (Single) and MH^4 do not consider AVL and FVL, and they assign a single voltage to each huddle. Therefore, “AVL” and “FVL” values of each huddle are the same.

The experimental results show that all energy consumption of AVHDR is reduced by 70.4% at the maximum and by 43.9% on average compared with the other algorithms. In SR [31], GDR [32], RDR [8], MH^4 (Single), and MH^4, dynamic multiple supply voltages can not been considered. Since AVHDR can dynamically assign voltages, SAAV achieved maximally 70.4% energy reduction. The energy

Table 5.6: Experimental results other than energy consumption.

App.	FUs	S_{max}	Architecture and algorithm	Rectangular area [μm^2]	i	CPU time [sec]
EWF3	Add×3	65	SR [31]	64664	2	53.97
	Mul×2		GDR [32]	42432	24	830.43
			RDR [8]	78400	1	105.35
			MH^4(Single)	48500	2	363.03
			MH^4	48360	5	632.65
			[6] + AVHDR	54766	2	378.86
			SAAV	52891	2	393.48
FIR	Add×4	30	SR [31]	74214	10	261.52
	Mul×4		GDR [32]	22165	24	2597.54
	Mem×1		RDR [8]	99225	1	191.02
			MH^4(Single)	31920	2	429.19
			MH^4	62238	4	1014.98
			[6] + AVHDR	60357	2	474.96
			SAAV	49440	5	1034.42
DCT	Add×4	15	SR [31]	62624	2	50.23
	Mul×4		GDR [32]	64925	24	1338.14
			RDR [8]	96800	1	191.41
			MH^4(Single)	63940	2	562.50
			MH^4	66766	4	919.89
			[6] + AVHDR	62040	2	575.40
			SAAV	68575	2	569.03
COPY	Add×3	175	SR [31]	440212	10	716.07
	Sub×1		GDR [32]	NA	–	NA
	Comp×1		RDR [8]	NA	–	NA
	Rshift×2		MH^4(Single)	433246	6	2179.72
	Mul×5		MH^4	396924	5	4178.51
	AND×1		[6] + AVHDR	415004	6	2936.31
	Mem×1		SAAV	384849	7	6160.19
PARKER	Add×2	10	SR [31]	18018	2	47.62
	Sub×2		GDR [32]	13464	2	92.68
	Comp×1		RDR [8]	20000	1	119.13
			MH^4(Single)	10080	2	299.40
			MH^4	11120	2	306.83
			[6] + AVHDR	11421	2	331.26
			SAAV	14852	2	309.34

consumption of AVHDR is reduced by 44.3% at the maximum and by 32.0% on the average compared with [6]. This is because the proposed algorithm can assign dynamic multiple supply voltages in scheduling/binding but [6] can assign voltages in only binding, which can provide more opportunities for assigning low voltage to non-critical operations or power gating steps.

5.6 Conclusion

In this chapter, I propose *an adaptive voltage huddle-based distributed-register architecture (AVHDR architecture)*, which integrates dynamic multiple supply voltages and interconnection delay into high-level synthesis. Next, I propose a high-level synthesis algorithm for *AVHDR architecture*. Experimental results show that the proposed algorithm achieves 43.9% energy-saving compared with conventional

Table 5.7: Energy comparison of experimental results.

App.	Architecture and algorithm	Average PG steps	Dynamic energy [pJ]	Leak energy [pJ]	Wire energy [pJ]	All energy [pJ]	Imp. [%]
EWF3	SR [31]	–	441.89	152.47	122.90	717.26	**57.8**
	GDR [32]	–	470.09	81.17	97.00	648.26	53.3
	RDR [8]	–	522.47	116.56	72.48	711.51	57.5
	MH^4(Single)	–	458.69	92.41	66.43	617.53	51.0
	MH^4	–	385.73	85.89	83.27	554.89	45.5
	[6] + AVHDR	6.0	400.65	90.35	52.57	543.56	44.3
	SAAV	17.4	225.84	49.87	26.98	302.69	–
FIR	SR [31]	–	347.10	100.06	55.95	503.10	**70.4**
	GDR [32]	–	210.50	17.75	51.80	280.06	46.8
	RDR [8]	–	250.17	27.37	82.30	359.84	58.6
	MH^4(Single)	–	210.17	27.50	38.01	275.67	46.0
	MH^4	–	203.13	30.74	41.70	275.57	46.0
	[6] + AVHDR	15.6	165.39	28.57	44.94	238.91	37.7
	SAAV	14.1	104.92	17.18	26.77	148.87	–
DCT	SR [31]	–	161.04	25.46	43.20	229.70	48.8
	GDR [32]	–	139.68	20.36	40.51	200.56	41.3
	RDR [8]	–	186.08	23.57	51.63	261.27	**55.0**
	MH^4(Single)	–	162.85	20.19	37.41	220.45	46.6
	MH^4	–	166.39	21.77	35.26	223.41	47.3
	[6] + AVHDR	1.8	128.02	18.55	25.16	171.73	31.5
	SAAV	4.4	83.08	12.98	21.64	117.70	–
COPY	SR [31]	–	8404.39	2534.78	853.88	11793.10	**67.3**
	GDR [32]	NA	NA	NA	NA	NA	NA
	RDR [8]	NA	NA	NA	NA	NA	NA
	MH^4(Single)	–	7742.40	1695.14	583.27	10020.80	61.6
	MH^4	–	3468.65	606.40	488.60	4563.65	15.6
	[6] + AVHDR	106.6	3377.95	591.73	570.11	4539.79	15.2
	SAAV	102.4	2840.98	517.82	492.62	3851.42	–
PARKER	SR [31]	–	38.07	3.65	14.42	56.14	50.4
	GDR [32]	–	28.42	2.57	33.67	64.67	**56.9**
	RDR [8]	–	30.91	2.41	3.87	37.20	25.1
	MH^4(Single)	–	24.66	2.48	5.90	33.04	15.7
	MH^4	–	21.67	1.33	6.53	29.54	5.7
	[6] + AVHDR	2.2	26.68	1.29	12.51	40.48	31.2
	SAAV	1.8	16.58	1.77	9.51	27.86	–

Table 5.8: Huddles configurations of MH^4 (Single) and Mh^4.

EWF3

MH^4 (Single)			MH^4		
FU (AVL[V])	#REG	FVL[V]	FU (AVL[V])	#REG	FVL[V]
Add (1.2)	5	1.2	Add (1.2)	6	1.2
Add (1.2)	5	1.2	Mul (1.2)		
Add (1.2)	1	1.2	Add (1.2)	2	1.2
Mul (1.2)	2	1.2	Mul (1.2)		
Mul (1.2)	2	1.2	Add (1.2)	5	1.2

FIR

MH^4 (Single)			MH^4		
FU (AVL[V])	#REG	FVL[V]	FU (AVL[V])	#REG	FVL[V]
Add (1.2)	3	1.2	Add (1.0)	3	1.0
Mul (1.2)			Mul (1.0)		
Mem (–)			Mul (1.2)	4	1.2
Add (1.2)	4	1.2	Mem (–)		
Add (1.2)			Add (1.0)	3	1.0
Add (1.2)	3	1.2	Add (1.0)	3	1.0
Mul (1.2)			Add (1.0)	2	1.0
Mul (1.2)	2	1.2	Mul (1.0)	2	1.0
Mul (1.2)	2	1.2	Mul (1.0)	2	1.0

DCT

MH^4 (Single)			MH^4		
FU (AVL[V])	#REG	FVL[V]	FU (AVL[V])	#REG	FVL[V]
Add (1.2)	5	1.2	Add (1.0)	5	1.0
Add (1.2)	3	1.2	Mul (1.0)		
Add (1.2)	3	1.2	Add (1.2)	5	1.2
Add (1.2)	3	1.2	Add (1.2)	5	1.2
Mul (1.2)	4	1.2	Add (1.0)	2	1.0
Mul (1.2)	4	1.2	Mul (1.2)	4	1.2
Mul (1.2)	3	1.2	Mul (1.0)	3	1.0
Mul (1.2)	3	1.2	Mul (1.0)	3	1.0

COPY

MH^4 (Single)			MH^4		
FU (AVL[V])	#REG	FVL[V]	FU (AVL[V])	#REG	FVL[V]
Add (1.2)	28	1.2	Add (0.8)	30	0.8
Comp (1.2)			Add (0.8)	15	0.8
Mem (–)			Add (0.8)	7	0.8
Add (1.2)	34	1.2	Sub (0.8)	19	0.8
Rshift (1.2)			Comp (1.0)	2	1.0
AND (1.2)	2	1.2	Rshift (1.0)	10	1.0
Mul (1.2)			Rshift (0.8)	9	0.8
Add (1.2)	0	1.2	Mul (0.8)	5	0.8
Sub (1.2)	26	1.2	Mul (0.8)	5	0.8
Rshift (1.2)	26	1.2	Mul (0.8)	4	0.8
Mul (1.2)	8	1.2	Mul (0.8)	4	0.8
Mul (1.2)	7	1.2	Mul (0.8)	4	0.8
Mul (1.2)	7	1.2	AND (0.8)	16	0.8
Mul (1.2)	6	1.2	Mem (–)	16	1.2

PARKER

MH^4 (Single)			MH^4		
FU (AVL[V])	#REG	FVL[V]	FU (AVL[V])	#REG	FVL[V]
Add (1.2)	5	1.2	Add (1.0)	4	1
Add (1.2)			Sub (1.0)		
Comp (1.2)			Comp (1.0)		
Sub (1.2)	3	1.2	Add (0.8)	4	0.8
Sub (1.2)	1	1.2	Sub (1.2)	2	1.2

Table 5.9: Huddles configurations of [6] + AVHDR and SAAV.

EWF3

[6] + AVHDR			SAAV		
FU (AVL[V])	#REG	FVL[V]	FU (AVL[V])	#REG	FVL[V]
Add (1.2, 1.0)	6	1.2	Add (1.2)	7	1.0
Add (1.2, 1.0)	6	1.2	Add (1.0, 0.8)	3	0.8
Add (0.8)	2	0.8	Add (1.0, 0.8)	3	0.8
Mul (1.0)	1	1.0	Mul (1.0, 0.8)	1	0.8
Mul (1.0)	1	1.0	Mul (1.2, 0.8)	1	0.8

FIR

[6] + AVHDR			SAAV		
FU (AVL[V])	#REG	FVL[V]	FU (AVL[V])	#REG	FVL[V]
Add (1.0)	6	1.0	Add (0.8)	4	0.8
Add (1.0, 0.8)			Mul (1.2, 0.8)		
Mul (1.0)			Mul (1.2, 1.0, 0.8)	3	0.8
Mul (1.2, 1.0)	2	1.2	Mul (1.2, 0.8)		
Mem (–)			Add (1.0, 0.8)	2	0.8
Mul (1.2, 1.0)	2	1.2	Add (1.0, 0.8)	2	0.8
Mul (1.2, 1.0)			Add (1.0, 0.8)	2	0.8
Add (1.2, 1.0, 0.8)	3	1.2	Mul (1.2, 0.8)	3	0.8
Add (0.8)	0	0.8	Mem (–)	0	1.2

DCT

[6] + AVHDR			SAAV		
FU (AVL[V])	#REG	FVL[V]	FU (AVL[V])	#REG	FVL[V]
Mul (1.0)	6	1.0	Add (1.2)	5	1.0
Mul (1.0)			Add (1.2, 1.0, 0.8)	3	0.8
Mul (1.0)	4	1.0	Add (1.0, 0.8)	3	0.8
Mul (1.0)			Add (1.0, 0.8)	3	0.8
Add (1.2, 1.0)	4	1.2	Mul (1.2, 0.8)	3	0.8
Add (1.2, 1.0)	3	1.2	Mul (1.0, 0.8)	3	0.8
Add (1.0, 0.8)	3	1.0	Mul (1.0, 0.8)	2	0.8
Add (1.0, 0.8)	3	1.0	Mul (1.0, 0.8)	2	0.8

COPY

[6] + AVHDR			SAAV		
FU (AVL[V])	#REG	FVL[V]	FU (AVL[V])	#REG	FVL[V]
Add (1.0, 0.8)	28	1.0	Mul (1.0, 0.8)	4	0.8
Add (0.8)	14	0.8	AND (0.8)		
Add (0.8)	13	0.8	Add (1.0, 0.8)	29	0.8
Sub (0.8)	18	0.8	Add (1.0, 0.8)	18	0.8
Comp (0.8)	2	0.8	Add (1.0, 0.8)	12	0.8
Rshift (1.0, 0.8)	27	1	Sub (0.8)	21	0.8
Rshift (0.8)	3	0.8	Comp (0.8)	2	0.8
Mul (0.8)	5	0.8	Rshift (1.0, 0.8)	8	1.0
Mul (0.8)	4	0.8	Rshift (0.8)	26	0.8
Mul (0.8)	4	0.8	Mul (1.0, 0.8)	5	0.8
Mul (0.8)	4	0.8	Mul (1.0, 0.8)	4	0.8
Mul (0.8)	4	0.8	Mul (0.8)	4	0.8
AND (0.8)	1	0.8	Mul (0.8)	4	0.8
Mem (–)	3	1.2	Mem (–)	3	1.2

PARKER

[6] + AVHDR			SAAV		
FU (AVL[V])	#REG	FVL[V]	FU (AVL[V])	#REG	FVL[V]
Add (1.0, 0.8)	4	1.0	Add (1.0, 0.8)	4	0.8
Sub (1.0, 0.8)			Sub (1.0, 0.8)		
Add (0.8)	4	0.8	Sub (1.2)	4	1.0
Sub (1.2, 0.8)	2	1.2	Comp (1.0, 0.8)		
Comp (1.0, 0.8)	3	1.0	Add (0.8)	1	0.8

algorithms.

Chapter 6

SAMCID: Multiple Clock Domains Aware High-level Synthesis Algorithm for HDR-mcd Architecture

6.1 Introduction

In this chapter, I first propose *an HDR-mcd architecture*, which integrates multiple clock domains and interconnection delay into high-level synthesis. In HDR-mcd, an entire chip is divided into several *huddles* whose sizes are determined by interconnection delay, and huddles are assumed to belong to clock domains. By doing this, HDR-mcd can realize synchronization between different clock domains in which interconnection delay is required and should be considered during high-level synthesis. Next, I propose a high-level synthesis algorithm for HDR-mcd, which can reduce energy consumption by optimizing configuration and placement of huddles. In the proposed iterative improvement based algorithm, low-frequency clocks are assigned to non-critical huddles under resource and latency constraints for energy efficiency improvement. Experimental results show that the proposed method achieves 32.5% energy-saving compared with the existing single clock domain based methods. Furthermore, the proposed method which can apply MCD and MSV simultaneously achieves 57.0% energy-saving compared with the conventional methods.

6.2 HDR-mcd Architecture

In this section, the conventional distributed-register architectures (DR architectures) and the conventional multiple clock domains techniques are briefly reviewed. Next, a new target architecture is proposed.

As devise feature size decreases, interconnection delays have become the dominant factor of the circuit total latency. Several DR architectures and synthesis algorithms [8, 17, 19, 31, 32] have been proposed in order to realize the HLS algorithms which consider not only the gate delays, such as the delays of functional units, but also the interconnection delays. In DR architectures, chip area is divided into sufficiently small partitions such that the interconnection delay inside each partition can be assumed to be zero. On the other hand, the inter-partition data transfer is realized through multi–cycle interconnect communication. The data transfer time of the multi-cycle interconnect communication is estimated by placement information which is obtained by partitions floorplanning during HLS. Several DR architectures, such as regular distributed-register architecture (RDR architecture) [8] and generalized distributed-register architecture (GDR architecture) [32] have been proposed. Their objectives are latency minimization and no energy-aware HLS algorithms for them have been proposed. For energy reduction, huddle-based distributed register architecture (HDR architecture) families with the corresponding synthesis algorithms were proposed. They utilized multiple supply voltages (MSV) (Chapters 3 and 4), clock gating [1], and power gating (PG) and/or dynamic multiple supply voltages (DMSV) (Chapter 5) in order to minimize energy consumption.

On the other hand, there are the techniques which can dynamically change clock frequency at run-time, such as DVFS [3]. However, there are no conventional high-level synthesis algorithms which consider DVFS. Scheduling algorithms and binding algorithms of high-level synthesis is based on a clock periods which are fixed at run-time. Therefore the problem definition which deal with frequency scaling is too difficult. F

lock scaled circuits. In this dissertation, different clock frequencies are going to be assigned to huddles which are the islands for the interconnection delay estimation of HDR architecture. In this case, the energy overheads of the synchronization circuits cannot be ignored. In this dissertation, the dynamically change of clock frequency is not considered as well as conventional high-level synthesis algorithms. This dissertation focuses on multiple clock domains (MCD) techniques and apply the MCD to the HDR architecture aiming at further energy reduction.

There are two types of MCD techniques: (1) One can utilize arbitrary clock frequencies but synchronization circuits are needed for the communication between

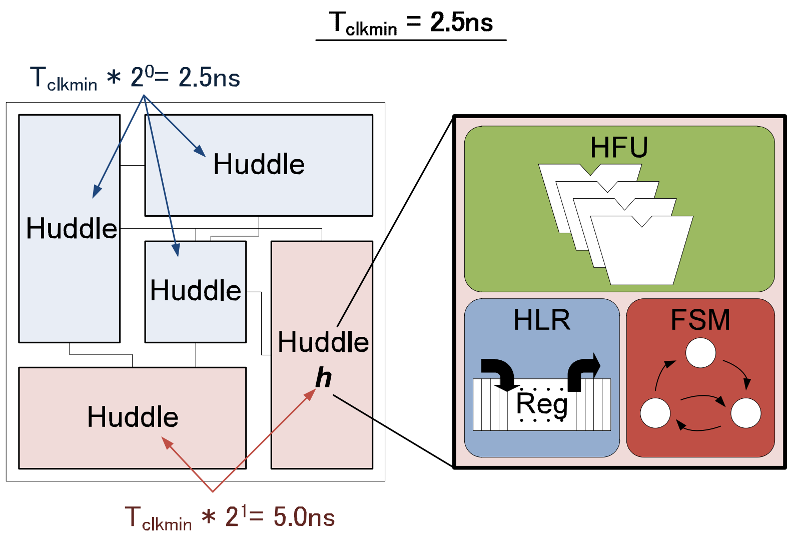


Figure 6.1: An HDR-mcd architecture.

different clock domains. (2) The other can communicate with different clock domains without synchronization circuits but can only utilize regular clock frequencies which follow certain fixed patterns. Global asynchronous local synchronous (GALS) [5] is well known as an example of Type (1). GLAS can reduce energy consumption effectively when optimal clocks are assigned to different cores in multiprocessor SoC or network-on-chip. On the other hand, GALS may increase energy consumption because of the synchronization circuit when GALS is applied to not so large circuits. In this dissertation, different clock frequencies are going to be assigned to huddles. In this case, the energy overheads of the synchronization circuits cannot be ignored because huddles are much smaller than processors or IP cores in order to ignore the interconnection delays inside huddles. Therefore, an MCD technique of Type (2) is applied. Periodically all-in-phase [30] is one of the MCD techniques of Type (2). In periodically all-in-phase technique, the clock periods of the local clock signals are the integer multiples of reference clock signals. Because the local clock signals can synchronize at the periods of common multiples, any synchronization circuits are not necessary.

However, original periodically all-in-phase cannot be applied to original HDR architecture. The reason is that periodically all-in-phase do not consider the interconnection delays between different clock domains. Furthermore, HDR architecture and synthesis algorithms assume that only one clock frequency is assigned to all huddles.

In order to apply periodically all-in-phase to HDR architecture, I propose a new distributed register architecture called *HDR-mcd architecture*. In the HDR-mcd architecture, chip areas are divided into several *huddles* which abstract functional units (FUs), registers, and controllers inside them. An HDR-mcd architecture example is shown in Fig. 6.1. A huddle h consists of the following components:

Huddled Local Registers (HLRs):
Dedicated local registers in h.

Huddled Functional Units (HFUs):
Dedicated functional units in h. HFUs can only access the HLRs in h.

Finite State Machine (FSM):
A dedicated controller in h. FSM controls the HFUs and the HLRs in h.

Clock frequencies are assigned to huddles and $T_{clk}(h)$ is the clock period which is assigned to huddle h. In HDR-mcd architecture, the selectable clock periods are limited to powers of 2 of the reference clock period. $T_{clk}(h)$ is calculated by $T_{clk}(h) = T_{clkmin} \cdot 2^n$ where T_{clkmin} is the reference clock period and n is an integer value greater than or equal to 0. We describe 2^n as a clock factor $CF(h)$ of a huddle h. T_{clk} of each huddle is determined during HLS process. The height and width of each huddle are limited by the assigned clock frequency value. This restriction is called *huddle size constraint*. Data transfer time inside huddles can be ignored because huddles are sufficiently small by introducing huddle size constraint. Interconnection delay between huddles is estimated by placement information which is obtained by huddle floorplanning during HLS.

Conventional HDR architectures and algorithms proposed in previous Chapters only consider single clock domain and manage only one huddle size constraint. However, because HDR-mcd architectures can deal with multiple clock domains, the huddle size constraint of each huddle is mutually different according to each assigned clock frequency. An appropriate huddle size constraint realizes suitable clock domain division and synchronization between different clock domains requiring interconnection delay.

6.3 Problem definition

Table 6.1 defines notations of the high-level synthesis problem targeting at HDR-mcd. In Table 6.1, $T_{clk}(h_j)$ shows the clock period which is assigned to huddle h_j and is calculated as follows using a clock factor $CF(h_j)$:

$$T_{clk}(h_j) = T_{clkmin} \cdot CF(h_j). \tag{6.1}$$

Table 6.1: Notation used in the HLS problem.

Term	Meaning
$G(N, E)$	A control data flow graph (CDFG) which is a directed graph.
N	A node set. $N = N_o \cup N_c$.
N_o	An operation node set.
N_c	A branching control node set.
	The branching control nodes are start and end nodes of conditional branches.
E	An edge set. $E = E_d \cup E_c$.
E_d	A data-flow edge set.
E_c	A control-flow edge set.
T_{clkmin}	A minimum clock period constraint.
S_{max}	A control step (CS) constraint.
s_t	A CS ($1 \leq s_t \leq S_{max}$).
F	A set of functional units (FUs).
f_i	An FU ($1 \leq i \leq \lvert F\rvert$).
$D_f(f_i)$	The delay of f_i.
n_x	An operation node ($1 \leq x \leq \lvert N_o\rvert$).
$Bind(n_x)$	The functional unit which executes n_x.
$OP(f_i)$	The set of operation nodes which are executed by f_i.
H	A set of Huddles.
h_j	A huddle ($1 \leq j \leq \lvert H\rvert \leq \lvert F\rvert$).
$Hud(f_i)$	The huddle where f_i is assigned.
$F(h_j)$	The set of FUs which are assigned to h_j.
$D_{reg}(h_j)$	The delay of the HLR in h_j.
$CF(h_j)$	The clock factor of h_j. $CF(h_j) = 2^n$
	where n is an integer value greater than or equal to 0.
$T_{clk}(h_j)$	The clock period which is assigned to h_j. Details can be found in Eq. (6.1).
$S_f(f_i)$	The number of CSs required to execute f_i. Details can be found in Eq. (6.2).
$Start(n_x)$	The CS when the execution of n_x starts.
$End(n_x)$	The CS when the execution of n_x ends. Details can be found in Eq. (6.3).
$Slack(f_i)$	The slack time of the FU f_i which can be used by data transfer
	for succeeding operations. Details can be found in Eq. (6.4).
$D_w(l)$	An interconnection delay whose length is l.
$W(h_j)$	The width of the huddle h_j.
$H(h_j)$	The height of the huddle h_j.
$Dist(h_j, h_k)$	The Manhattan distance between the huddles h_j and h_k.
$Tr(f_i, h_k)$	The inter-huddle data transfer delay from f_i to HLRs in h_k.
	Details can be found in Eq. (6.6).
$DT(f_i, h_k)$	The number of clock cycles required to transfer data from f_i to h_k.
	Details can be found in Eq. (6.7).
DT	The data transfer table which is defined by a $\lvert F\rvert \times \lvert H\rvert$ matrix
	whose (i, k)-element is expressed by $DT(f_i, h_k)$.
e_y	A data-flow edge ($1 \leq y \leq \lvert E_d\rvert$).
$Start(e_y)$	The CS when the data transfer of e_y starts.
$End(e_y)$	The CS when the data transfer of e_y ends. Details can be found in Eq. (6.8).

$S_f(f_i)$ shows the number of control steps (CSs) required to execute f_i. In the case of $f_i \in F(h_j)$, $S_f(f_i)$ is calculated as follows:

$$S_f(f_i) = \left\lceil \frac{D_f(f_i) + D_{reg}(h_j)}{T_{clk}(h_j)} \right\rceil \cdot CF(h_j). \tag{6.2}$$

$End(n_x)$ is the CS when the execution of n_x ends and calculated as follows:

$$End(n_x) = Start(n_x) + S_f(f_i) - 1. \tag{6.3}$$

$Slack(f_i)$ is defined by:

$$Slack(f_i) = T_{clkmin} \cdot S_f(f_i) - D_f(f_i). \tag{6.4}$$

$Slack(f_i)$ shows the slack time of the FU f_i which can be used by data transfer for succeeding operations. The width and height of each huddle must satisfy the following *huddle size constraint*:

$$2 \cdot D_w(W(h_j) + H(h_j)) \leq \min_{f_i \in F(h_j)} \{Slack(f_i)\}, \tag{6.5}$$

where $W(h_j)$ and $H(h_j)$ are the width and height of the huddle h_j, respectively. $2 \cdot D_w(W(h_j) + H(h_j))$ shows the maximum interconnection delay for the intra-huddle communication inside the huddle h_j. In the proposed algorithm, we obtain the value of $(W(h_j) + H(h_j))$ so that it satisfies the huddle size constraint and determine $W(h_j)$ and $H(h_j)$ by using the aspect ratio predefined for each huddle.

$Tr(f_i, h_k)$ shows the inter-huddle data transfer delay from $f_i (\in F(h_j))$ to HLRs in h_k which is defined by:

$$Tr(f_i, h_k) = D_w(Dist(h_j, h_k)) + D_{reg}(h_k). \tag{6.6}$$

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

$$DT(f_i, h_k) = \begin{cases} 0 \quad (Slack(f_i) \geq Tr(f_i, h_k)), \\ \\ \lceil Tr(f_i, h_k)/T_{clkmin} \rceil \\ \qquad (Slack(f_i) < Tr(f_i, h_k)). \end{cases} \tag{6.7}$$

We prepare two types of data transfer mode:

Mode 1: In the case of $Slack(f_i) \geq Tr(f_i, h_k)$, the functional unit $f_i \in F(h_j)$ directly stores its output into the registers in the huddle h_k. Thus, the data transfer requires no extra cycles.

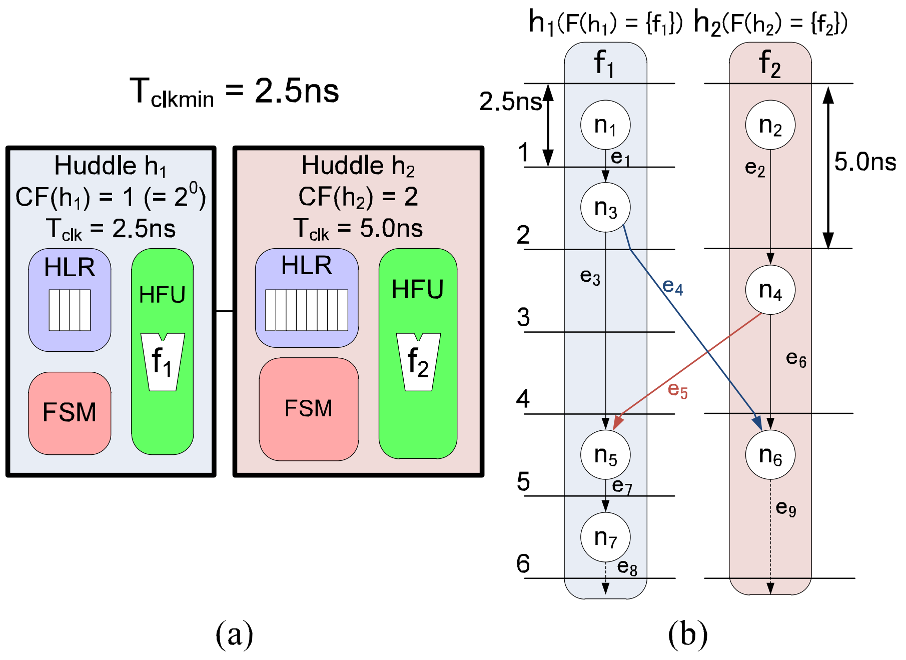


Figure 6.2: An HLS result for HDR-mcd. (a) An HDR-mcd configuration. (b) A scheduled DFG.

Mode 2: In the case of $Slack(f_i) < Tr(f_i, h_k)$, the functional unit $f_i \in F(h_j)$ first stores its output into the registers in the huddle h_j. In the subsequent cycles, we perform the data transfer from the huddle h_j to the huddle h_k. The data transfer requires $\lceil Tr(f_i, h_k)/T_{clk} \rceil$ cycles.

$Start(e_y)$ is the CS when the data transfer of e_y starts. If e_y is one of the outputs of n_x, $Start(e_y) = End(n_x)$. $End(e_y)$ is the CS when the data transfer of e_y ends. When e_y is the data-flow from f_i to h_k, $End(e_y)$ is calculated as follows:

$$End(e_y) = \left\lceil \frac{Start(e_y) + DT(f_i, h_k)}{CF(h_k)} \right\rceil \cdot CF(h_k). \tag{6.8}$$

Example 6.1. *Fig. 6.2 shows an example of HDR-mcd architecture when minimum clock period constraint* $T_{clkmin} = 2.5\,\text{ns}$ *and a CS constraint* $S_{max} = 6$ *are given.*

In this example, we have huddle configurations of $F(h_1) = \{f_1\}$ *and* $F(h_2) = \{f_2\}$*. Clock factors are assigned to the huddles as in Fig. 6.2:* $CF(h_1) = 1 (= 2^0)$ *and* $CF(h_2) = 2$*. Therefore,* $T_{clk}(h_1) = 2.5\,\text{ns}$ *and* $T_{clk}(h_2) = 5.0\,\text{ns}$ *are assigned to the huddles.*

Let $D_f(f_1) = D_f(f_2) = 1.5\,\text{ns}$ *and* $D_{reg}(h_1) = D_{reg}(h_2) = 0.5\,\text{ns}$.
$D_w(Dist(h_1, h_2)) = 1.0\,\text{ns}$ *is the interconnection delay between* h_1 *and* h_2*.* $S_f(f_1)$

and $S_f(f_2)$ can be calculated by:

$$S_f(f_1) = \lceil (1.5\,\mathrm{ns} + 0.5\,\mathrm{ns})/2.5\,\mathrm{ns} \rceil \cdot 1 = 1,$$
$$S_f(f_2) = \lceil (1.5\,\mathrm{ns} + 0.5\,\mathrm{ns})/5.0\,\mathrm{ns} \rceil \cdot 2 = 2.$$

$Slack(f_1)$ and $Slack(f_2)$ can be calculated by:

$$Slack(f_1) = 2.5\,\mathrm{ns} \cdot 1 - 1.5\,\mathrm{ns} = 1.0\,\mathrm{ns},$$
$$Slack(f_2) = 2.5\,\mathrm{ns} \cdot 2 - 1.5\,\mathrm{ns} = 3.5\,\mathrm{ns}.$$

$Tr(f_1, h_2)$ and $Tr(f_2, h_1)$ are calculated by:

$$Tr(f_1, h_2) = Tr(f_2, h_1) = 1.0\,\mathrm{ns} + 0.5\,\mathrm{ns} = 1.5\,\mathrm{ns}.$$

Since $Slack(f_2) \geq Tr(f_2, h_1)$, the data transfer from f_2 to h_1 is based on Mode 1 and $DT(f_2, h_1)$ is calculated by:

$$DT(f_2, h_1) = 0.$$

We focus on the edge e_5. As in Fig. 6.2, the start CS of the data transfer is $Start(e_5) = End(n_4) = 4$. Based on $Start(e_5)$ and $DT(f_2, h_1)$, $End(e_5)$ is calculated by:

$$End(e_5) = \lceil (4 + 0)/1 \rceil \cdot 1 = 4.$$

Because the data transfer e_5 can be completed in CS 4, the succeeding node n_5 can be executed from CS 5. On the other hand, since $Slack(f_2) < Tr(f_2, h_1)$, the data transfer from f_1 to h_2 is based on Mode 2 and $DT(f_1, h_2)$ is calculated by:

$$DT(f_1, h_2) = \lceil 1.5/2.5 \rceil = 1.$$

We focus on the edge e_4. As in Fig. 6.2, the start CS of the data transfer is $Start(e_4) = End(n_3) = 2$. Based on $Start(e_4)$ and $DT(f_1, h_2)$, $End(e_4)$ is calculated by:

$$End(e_4) = \lceil (2 + 1)/2 \rceil \cdot 2 = 4.$$

Because $T_{clk}(h_2) = 5.0\,\mathrm{ns}$ is assigned to h_2, the data transfer of e_4 cannot be completed in CS 3 but the data is stored at the clock edge of CS 4. □

Based on the above definitions, the HLS problem is defined as follows:

Definition 6.1. *The HLS problem is, for a given CDFG, a minimum clock cycle constraint, a CS constraint, and a set of functional units, to assign each operation node to a CS and a functional unit to bind each functional unit to each huddle, to bind clock period to each huddle, so that the given CDFG is executed correctly considering multi-cycle interconnect communications and periodically all-in-phase based MCD. The objective is to minimize the total energy consumption.* □

6.4 The SAMCID Algorithm

In this section, a new high-level Synthesis Algorithm considering Multiple Clock domains and Interconnection Dela   lled *SAMCID* is proposed.

Generally, high-level synthesis algorithms considering component placement simultaneously are composed of schedulings, bindings, and floorplannings and classified into the following two types:

Type 1: Schedulings, bindings, and floorplannings are executed a predetermined number of times in a predetermined order.

Type 2: Schedulings, bindings, and floorplannings are executed repeatedly as an iterative refinement flow.

In Type 1, a required computation time to synthesize a chip can be expected since the number of executed synthesis steps and the execution order are determined. If we know in advance how many iterations we need to perform each high-level synthesis step as well as its best execution order, Type 1 will be the best choice. MCAS [8], one of the RDR architecture synthesis algorithms, uses an approach based on Type 1 above.

In Type 2, required informations such as scheduling and placement results are fed back to each other since each synthesis step is executed repeatedly to obtain the optimal solution. The GDR architecture synthesis algorithm [32], the HDR architecture synthesis algorithms and SAAV proposed in previous Chaptersuse approaches based on Type 2. By iteratively executing scheduling/binding steps and floorplanning steps, the current scheduling/binding step can consider interconnection dela   lanning step. The shape and size of each module are determined in a scheduling/binding step and a floorplanning step is done using these module informations. Because each synthesis step affects each other, iterative refinement flows as in Type 2 is the best choice targeting GDR, HDR, and AVHDR.

In SAMCID, we consider the interconnection delay effects and the synchronization between different clock domains. It is very difficult to predict the influence of these components in algorithms based on Type 1. Therefore Type 2-based synthesis flow is accepted for the proposed HDR-mcd architecture, in which global optimization could be achieved by considering multiple clock domains.

Based on Type 2, a virtual-area-based iterative refinement flow is used in SAMCID. We prepare two types of area, called a real area and a virtual area. Let $A_{real}(h_j)$ be the real area of the huddle h_j, and $A_{real}(h_j)$ is the sum of functional unit areas and register areas inside h_j. Let $A_{virtual}$ be the virtual area of huddle

h_j, and $A_{virtual}(h_j)$ is estimated in the iteration when its huddle construction is changed as follows:

1. In each iteration, $A_{real}(h_j)$ is calculated by summing up the areas of functional units, registers, a controller, and level converters inside h_j.
2. If $A_{virtual}(h_j) \geq A_{real}(h_j)$, $A_{virtual}(h_j)$ is not updated.
3. If $A_{virtual}(h_j) < A_{real}(h_j)$, we set
   $A_{virtual}(h_j) = A_{real}(h_j)$.

The synthesis algorithm is mainly composed of three processes: initial process, iteration process, and adjustment process. In initial and adjustment process, huddle placement is determined based on the real area of each huddle. In iteration process, we perform scheduling/binding and floorplanning repeatedly based on virtual area of each huddle. When no timing violations occur in the iteration process, we go to the adjustment process.

SAMCID is composed of the following seven steps in each iteration:

- **initial huddling,**
- **scheduling/binding,**
- **register/controller synthesis,**
- **unhuddling,**
- **floorplanning,**
- **floorplanning-directed huddling,**
- **virtual area adaptation.**

In the initial huddling step, the initial huddle configuration and the placement are determined by given functional units. In the scheduling/binding step, each operation node in a CDFG is assigned to a CS and a FU and each huddle is assigned a corresponding clock period considering multiple clock domains and multi-cycle interconnect communications. In the register/controller synthesis step, HLR and FSM configurations in each huddle are determined using the scheduling/binding result. In unhuddling step, we divide huddles which violate the huddle size constraint. In the floorplanning step, every huddle is placed. In the floorplanning-directed huddling step, the configuration and placement of each huddle is determined simultaneously. In the virtual area adaptation step, we reduce the virtual area overhead during iteration. Fig. 6.3 shows the SAMCID algorithm. In the rest of this section, we will describe the the proposed SAMCID in details. Note that we deal with only DFG and only one operation type for simplicity as a motivated example but we can deal with CDFG and any other operation type similarly.

6.4.1 Initial huddling

In initial huddling, initial huddle configuration and placement are determined according to the given functional units. If a set of functional units F is given as an input, we prepare $|F|$ huddles. Each functional unit is assigned to each huddle and the minimum clock cycle T_{clkmin} is assigned to each huddle. All the huddles are overlapped with each other where we can ignore interconnection delay between huddles here.

Example 6.2. *In initial huddling, initial huddle configuration and placement are determine by a given set of functional units F. The input and the output are as follows:*

Input *:*

- $F = \{f_1, f_2, f_3, f_4\}$.

Output *:*

- *huddle configuration (Fig. 6.4).*

Because all the huddles are overlapped with each other, we can ignore the interconnection delay between huddles in the initial process. □

6.4.2 Scheduling/binding

The scheduling/binding problem here is, for given a CDFG $G(N, E)$, a minimum clock period constraint T_{clkmin}, a control step constraint S_{max}, a set of functional units F, and huddle configuration, to find scheduling and functional unit binding of every node in a given CDFG and to determine clock periods $T_{clk}(h_j)$ of each huddle h_j so as to minimize the total energy consumption while meeting the control step constraint.

The scheduling/binding is composed of the three phases: initial phase, clock period assignment phase, and operation scheduling/binding phase.

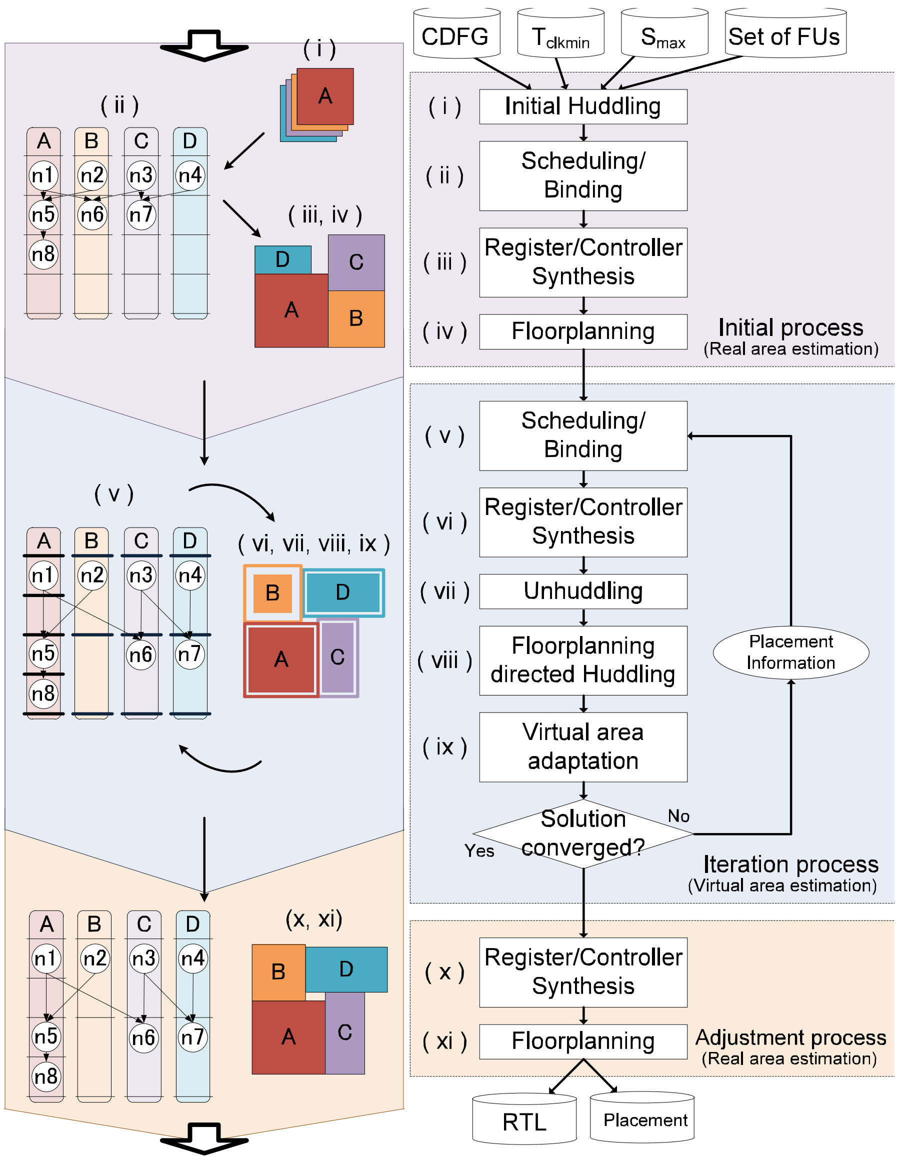


Figure 6.3: Proposed synthesis algorithm called SAMCID.

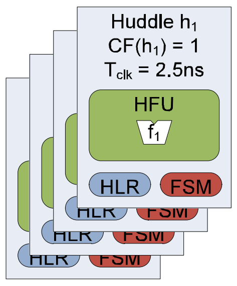


Figure 6.4: The result of the initial huddling.

(1) Initial phase:
In the initial phase, scheduling and binding are executed according to huddle placement and clock periods obtained by the previous iteration.

(2) Clock period assignment phase:
In the clock period assignment phase, we search the clock periods assignment so as to minimize total energy consumption.

(3) Operation scheduling/binding phase:
In the operation scheduling/binding phase, operations are assigned to control steps and FUs so as to minimize total energy consumption.

All these details are explained in the following paragraphs.

The basic scheduling/binding algorithm, which is used in all three phases, is proposed. This proposed basic scheduling/binding algorithm is based on the data-transfer-table based list scheduling algorithm [31]. The basic scheduling/binding picks up the most critical node one by one and assigns it to a CS and an FU so that the total number of expected CSs is minimized. Critical path length $CP(n_x, f_j)$ refers to the longest path length from n_x to any end node if n_x is assigned to an FU f_j. $CP(n_x, f_j)$ can be calculated by:

$$CP(n_x, f_j) = S_f(f_j) + \max_{n_k \in succ(n_x)} S_{init}(n_k, f_j), \tag{6.9}$$

where $succ(n_x)$ is a set of immediate successors of n_x and $S_{init}(n_k, f_j)$ is the CS when f_j become available for n_k execution. $S_{init}(n_k, f_j)$ is calculated as follows:

$$S_{init}(n_k, f_j) = \min_{f_l \in F} \left\{ DT(f_l, Hud(f_j)) + CP(n_k, f_l) \right\}. \tag{6.10}$$

We can calculate $CP(n_x, f_j)$ for each node n_x and each FU f_j by calculating Eq. (6.9) and Eq. (6.10) from an end node to start node. Based on the critical

Algorithm 6.1 Basic scheduling/binding algorithm.

```
1: Calculate $DT(f_j, h_k)$ for each FU $f_j$ and each huddle $h_k$.
2: $N_{target} \leftarrow N_o$.
3: Calculate $CP(n_x, f_j)$ for each operation node $n_x$ and each FU $f_j$.
4: Calculate $P_{list}(n_x)$ for each operation node $n_x$.
5: $CS_{trgt} \leftarrow 0$.
6: while $N_{target} \neq \phi$ do
7:    $CS_{trgt} \leftarrow CS_{trgt} + 1$.
8:    Let $N_{ready}$ be a set of nodes whose successors are already scheduled.
9:    while $N_{ready} \neq \phi$ do
10:      Pick up the node $n_x \in N_{ready}$ whose $P_{list}(n_x)$ is the maximum.
11:      Calculate $L(n_x, f_j)$ for each FU $f_j$ which can execute $n_x$.
12:      Pick up the FU $f_j$ whose $L(n_x, f_j)$ is the minimum.
13:      if $CS_{trgt} > \max_{n_l \in pred(n_x)} \left\{ \left\lceil \frac{End(n_l) + DT(f_l, h_j)}{CF(h_j)} \right\rceil \cdot CF(h_j) \right\}$ then
14:         $Bind(n_x) \leftarrow f_j$, $Start(n_x) \leftarrow CS_{trgt}$.
15:         $N_{target} \leftarrow N_{target} - \{n_x\}$.
16:      end if
17:      $N_{ready} \leftarrow N_{ready} - \{n_x\}$.
18:   end while
19: end while
```

path length, let $P_{list}(n_x)$ be the priority function of the list scheduling and $P_{list}(n_x)$ is calculated as follows:

$$P_{list}(n_x) = \min_{f_l \in F} CP(n_k, f_l), \tag{6.11}$$

Then from a start node to an end node, we pick up a node n_x whose $P_{list}(n_x)$ is the maximum. After that, we bind n_x to the FU f_j so that estimated latency $L(n_x, f_j)$ is minimized. Estimated latency $L(n_x, f_j)$ is calculated as:

$$L(n_i, f_j) = \max_{n_x \in pred(n_x)} \left\{ \left\lceil \frac{End(n_l) + DT(f_l, h_j)}{CF(h_j)} \right\rceil \cdot CF(h_j) \right\} + CP(n_x, f_j), \tag{6.12}$$

where $pred(n_x)$ is a set of immediate predecessors of n_x, $f_l = Bind(n_l)$, and $h_j = Hud(f_j)$. Algorithm 6.1 shows the proposed basic scheduling/binding algorithm.

Initial phase

In the initial phase, scheduling and binding are executed according to huddle placement and clock periods obtained by the previous iteration. If the scheduling

Algorithm 6.2 Initial phase.

```
1: Execute basic scheduling/binding (Algorithm 6.1).
2: if the result satisfy $S_{max}$ then
3:    End.
4: end if
5: Calculate $P_s(h_j)$ for each huddle $h_j$.
6: while  do
7:    for $h_j$ in the increasing-order of $P_s(h_j)$ do
8:       if $CF(h_j) = 1$ then
9:          Go to next step 7.
10:      end if
11:      $CF(h_j) \leftarrow CF(h_j)/2$ and execute basic scheduling/binding.
12:      if the result satisfy $S_{max}$ then
13:         End.
14:      end if
15:   end for
16:   if $CF(h_j) = 1$ for all $h_j$ then
17:      End.
18:   end if
19: end while
```

result based on the previous iteration does not satisfy the control step constraint, clock periods are decreased. In order to decrease the clock periods, we design a priority $P_s(h_j)$ for a huddle h_j. $P_s(h_j)$ is calculated by:

$$P_s(h_j) = \sum_{f_i \in F(h_j)} |OP(f_i)|. \tag{6.13}$$

We pick up the huddle whose priority $P_s(h_j)$ is the smallest first and try to decrease its $CF(h_j)$. We repeat this process unless a scheduled CDFG satisfy the CS constraint S_{max}. In (ii) scheduling/binding of initial process, only (1) initial phase is done and interconnection delays are ignored since we assume that all the huddles are overlapped with each other. Algorithm 6.2 shows the initial process.

Clock period assignment phase

In Clock period assignment phase, we search the clock periods assignment. So as to minimize total energy consumption, we design a cost function $Cost_s$. $Cost_s$ is the sum of the energy consumption of registers and is calculated as:

$$Cost_s = \sum_{\forall h_j} \big|R(h_j)\big| \cdot \Big\{R_d S_{result}/CF(h_j) + R_l S_{result} T_{clkmin}\Big\}, \tag{6.14}$$

where $R(h_j)$ is the set of registers in h_j, R_d is a dynamic energy consumption of a register, R_l is a leak power of a register, and S_{result} is the end CS of the scheduling result ($S_{result} \leq S_{max}$). $R(h_j)$ of each huddle is estimated by left-edge based register binding algorithm. In scheduling/binding, we cannot calculate total energy consumption accurately since huddle placements are not determined. In this step, we can calculate the energy consumption of registers and the energy consumption of them finally occupies most of the circuit total energy consumption. The energy consumption of the clock trees also occupies most of it. The accurate energy of clock trees is calculated from the number of registers and the placement of registers. In scheduling/binding, the placement of them remains unknown but the number of them can be calculated. Therefore, the energy consumption of registers is a reasonable cost in this step.

A nexus huddle set H_{nex} and a maverick huddle h_{mav} are proposed. First, H_{nex} is a set of the all huddles which are assigned to a target clock factor CF_{trgt}. We tried to change clock factors of all huddles in H_{nex} from CF_{trgt} to $2 \cdot CF_{trgt}$ and basic scheduling/binding is done. If the result violates the CS constraint S_{max}, we pick up a maverick huddle h_{mav} in H_{nex} using the following priority $P_{mav}(h_i)$. In order to pick up a huddle which is arranged at a position farthest from the H_{nex}, we design a priority $P_{mav}(h_i)$ for a huddle h_i. $P_{mav}(h_i)$ is calculated by:

$$P_{mav}(h_i) = W(H_{nex} - \{h_i\}) + H(H_{nex} - \{h_i\}), \tag{6.15}$$

where $W(H)$ and $H(H)$ mean the width and height of the smallest bounding rectangle of a set of huddle H. We calculate $P_{mav}(h_i)$ for each huddle $h_i \in H_{nex}$. We pick up a huddle h_i whose $P_{mav}(h_i)$ is the minimum and h_i is removed from H_{nex} as a maverick huddle. Then, we tried to change the clock factors of H_{nex} and h_{mav} separately. We repeat this process unless a scheduled CDFG satisfy the CS constraint S_{max}. If the scheduling result satisfy S_{max}, we let $CF_{trgt} = 2 \cdot CF_{trgt}$ and repeat the processes. Finally, we get the clock period assignment solution whose $Cost_s$ is the minimum. Algorithm 6.3 shows the clock period assignment phase.

Operation scheduling/binding phase

In the operation scheduling/binding phase, operations are assigned to control steps and FUs so as to minimize total energy consumption. We use the same cost function $Cost_s$ as the clock period assignment phase. Algorithm 6.4 shows the operation scheduling/binding phase.

Example 6.3. *This example shows that the step (v) just after the initial process.*

In scheduling/binding, operation scheduling and functional unit binding are executed. The inputs and the outputs are as follows:

Algorithm 6.3 Clock period assignment phase.

```
1: Calculate $Cost_s$
2: $MinCost_s \leftarrow Cost_s$.
3: $CF_{trgt} \leftarrow 1$
4: while $CF_{trgt} < S_{max}$ do
5:    Let $H_{nex} = \{h_j | CF(h_j) = CF_{trgt}\}$.
6:    $h_{mav} = \phi$, $FLAG \leftarrow false$.
7:    $CF(h_j) \leftarrow 2 \cdot CF_{trgt}$ for all $H_{nex}$.
8:    Execute basic scheduling/binding (Algorithm 6.1).
9:    if the result violate the CS constraint $S_{max}$ then
10:      $CF(h_j) \leftarrow CF_{trgt}$ for all $H_{nex}$.
11:   else
12:      $FLAG \leftarrow true$.
13:      Calculate $Cost_s$.
14:      if $MinCost_s > Cost_s$ then
15:         $MinCost_s \leftarrow Cost_s$.
16:      end if
17:   end if
18:   if $h_{mav} \neq \phi$ then
19:      $CF(h_{mav}) \leftarrow 2 \cdot CF_{trgt}$
20:      Execute basic scheduling/binding (Algorithm 6.1).
21:      if the result violates the CS constraint $S_{max}$ then
22:         $CF(h_{mav}) \leftarrow CF_{trgt}$.
23:      else
24:         Calculate $Cost_s$.
25:         if $MinCost_s > Cost_s$ then
26:            $MinCost_s \leftarrow Cost_s$.
27:         end if
28:      end if
29:   end if
30:   if FLAG then
31:      $CF_{trgt} \leftarrow 2 \cdot CF_{trgt}$
32:      Go to step 5.
33:   end if
34:   Calculate $P_{mav}(h_j)$ for each huddle $h_j \in H_{nex}$.
35:   Pick up $h_j$ whose $P_{mav}(h_j)$ is the minimum, $h_{mav} \leftarrow h_j$.
36:   $H_{nex} \leftarrow H_{nex} - \{h_j\}$
37:   Go to step 7.
38: end while
39: Return the clock period assignment of $Mincost_s$.
```

Algorithm 6.4 Operation scheduling/binding phase.

```
Require: $Bind(n_x)$ of the clock period assignment phase.
 1: Calculate $Cost_s$, $Mincost_s \leftarrow Cost_s$.
 2: Calculate $P_{list}(n_x)$ for each operation node $n_x$.
 3: for $n_x$ in the increasing-order of $P_{list}(n_x)$ do
 4:   $MinCostFU \leftarrow Bind(n_x)$
 5:   for $\forall f_i$ which can execute $n_x$ do
 6:     $Bind(n_x) \leftarrow f_i$.
 7:     Execute basic scheduling/binding (Algorithm 6.1) without changing
        $Bind(n_x)$ of all operation nodes.
 8:     Calculate $Cost_s$.
 9:     if The result satisfies the CS constraint $S_{max}$ and $MinCost_s > Cost_s$
        then
10:       $MinCost_s \leftarrow Cost_s$.
11:       $MinCostFU \leftarrow f_i$
12:     end if
13:   end for
14:   $Bind(n_x) \leftarrow MinCostFU$
15: end for
16: Return scheduling/binding result of $MinCost_s$.
```

Inputs *:*

- *DFG $G(N, E)$ (Fig. 6.5(a)),*
- $D_f(f_1) = D_f(f_2) = D_f(f_3) = D_f(f_4) = 1.5\,\text{ns}$,
- *Register information (Table 6.2),*
- $T_{clkmin} = 2.5\,\text{ns}$,
- $S_{max} = 6$, *and*
- *huddle configuration (Fig. 6.5(b)).*

Outputs *:*

- *scheduled DFG (Fig. 6.11(a)) and*
- *clock assignment (Fig. 6.11(b)).*

First, we assume interconnection delay between huddles. According to the example showing in Fig 6.5(b), the interconnection delay can be assumed as follows:

$$D_w(Dist(h_1, h_2)) = D_w(Dist(h_1, h_3))$$
$$= D_w(Dist(h_2, h_4)) = D_w(Dist(h_3, h_4)) = 1.0\,\text{ns},$$

$$D_w(Dist(h_1, h_4)) = D_w(Dist(h_2, h_3)) = 2.0\,\text{ns}.$$

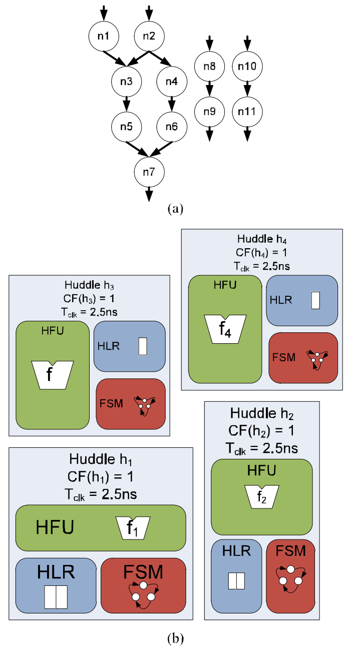


Figure 6.5: The inputs of scheduling/binding.

Table 6.2: The register information.

	D_{reg}	R_d	R_l
REG	0.5 ns	200 fJ	1.5 μW

We go through initial process.

Initial phase: *First, basic scheduling/binding (Algorithm 6.1) is executed. We can calculate $S_f(f_1)$ as follows:*

$$\begin{aligned} S_f(f_1) &= \lceil (D_f(f_1) + D_{reg}(h_1))/T_{clk} \rceil \cdot CF(h_1) \\ &= \lceil (1.5\,\text{ns} + 0.5\,\text{ns})/2.5\,\text{ns} \rceil \cdot 1 \\ &= 1 (= S_f(f_2) = S_f(f_3) = S_f(f_4)). \end{aligned}$$

The critical path length $CP(n_x, f_j)$ for each n_x and each f_i is calculated from end nodes. In this example n_7, n_9, and n_{11} are the end nodes. First, we calculate $CP(n_7, f_1)$, $CP(n_9, f_1)$, and $CP(n_{11}, f_1)$ as follows:

$$\begin{aligned} CP(n_7, f_1) &= S_f(f_1) = 1 \\ &= CP(n_9, f_1) = CP(n_{11}, f_1). \end{aligned}$$

Next, we calculate the critical path length of the predecessors of the end nodes. $CP(n_5, f_1)$ is calculated as follows:

$$\begin{aligned} CP(n_5, f_1) &= S_f(f_1) + \max_{n_k \in succ(n_5)} S_{init}(n_k, f_1) \\ &= 1 + \max_{n_7 \in succ(n_5)} S_{init}(n_7, f_1). \end{aligned}$$

Then, $S_{init}(n_7, f_1)$ can calculated as follows:

$$\begin{aligned} S_{init}(n_7, f_1) &= \min_{f_l \in F}\{DT(f_l, Hud(f_1)) + CP(n_5, f_l)\} \\ &= \min\{DT(f_1, h_1) + CP(n_5, f_1), \\ &\qquad DT(f_2, h_1) + CP(n_5, f_2), \\ &\qquad DT(f_3, h_1) + CP(n_5, f_3), \\ &\qquad DT(f_3, h_1) + CP(n_5, f_3)\} \\ &= \min\{0 + 1, 1 + 1, 1 + 1, 1 + 1\} \\ &= 1. \end{aligned}$$

Therefore, $CP(n_5, f_1)$ is finally calculated as follows:

$$CP(n_5, f_1) = 1 + 1 = 2.$$

Similarly, $CP(n_x, f_j)$ for each n_x and each f_i is calculated and Table 6.3 shows the $CP(n_x, f_j)$. At $CS_{trgt} = 1$, $N_{ready} = \{n_1, n_2, n_8, n_{10}\}$ and we pick up n_1. We calculate $L(n_1, f_j)$ for each f_j as follows:

$$\begin{aligned} L(n_1, f_1) &= CP(n_1, f_1) = 1 \\ &\qquad = L(n_1, f_2) = L(n_1, f_3) = L(n_1, f_4). \end{aligned}$$

Therefore, $Bind(n_1) = f_1$, $Start(n_1) = 1$. Similarly, we schedule and bind each node and Fig. 6.6(a) shows the result of basic scheduling/binding. Since the result satisfy the CS constraint S_{max}, we finish the initial phase.

Clock period assignment phase: *First, we calculate $Cost_s$. Based on the result of the initial phase, $|R(h_1)| = 2$, $|R(h_2)| = 2$, $|R(h_3)| = 1$, and $|R(h_4)| = 1$. Therefore, $Cost_s$ is calculated as follows:*

Table 6.3: $CP(n_x, f_j)$ for each n_x and each f_i.

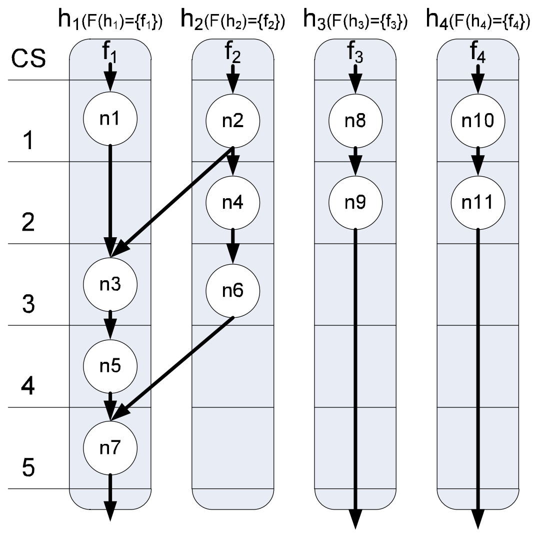
	f_1	f_2	f_3	f_4
n_1	**4**	4	4	4
n_2	**4**	4	4	4
n_3	**3**	3	3	3
n_4	**3**	3	3	3
n_5	**2**	2	2	2
n_6	**2**	2	2	2
n_7	**1**	1	1	1
n_8	**2**	2	2	2
n_9	**1**	1	1	1
n_{10}	**2**	2	2	2
n_{11}	**1**	1	1	1

$$\begin{aligned} Cost_s = \sum_{\forall h_j} |R(h_j)| \cdot \{R_d S_{result}/CF(h_j) + R_l S_{result} T_{clkmin}\} \\ = 2 \cdot (200\,\text{fJ} \cdot 5/1 + 1.5\,\mu\text{W} \cdot 5 \cdot 2.5\,\text{ns}) \\ +2 \cdot (200\,\text{fJ} \cdot 5/1 + 1.5\,\mu\text{W} \cdot 5 \cdot 2.5\,\text{ns}) \\ +1 \cdot (200\,\text{fJ} \cdot 5/1 + 1.5\,\mu\text{W} \cdot 5 \cdot 2.5\,\text{ns}) \\ +1 \cdot (200\,\text{fJ} \cdot 5/1 + 1.5\,\mu\text{W} \cdot 5 \cdot 2.5\,\text{ns}) \\ = 6112.5\,\text{fJ}. \end{aligned}$$

We let $H_{nex} = \{h_1, h_2, h_3, h_4\}$ *and assign* $T_{clk}(h_j) = 2.5\,\text{ns} \cdot 2 = 5.0\,\text{ns}$ *to each huddle* $h_j \in H_{nex}$*. Based on this clock assignment, we try to execute basic scheduling/binding. Fig. 6.7 shows the results of the basic scheduling/binding and associated data transfer table. Since the results violate the CS constraint,* $T_{clk}(h_j) = 2.5\,\text{ns}$ *is reassigned to each huddle* $h_j \in H_{nex}$*.*

Then we calculate $P_{mav}(h_j)$ *for each huddle* $h_j \in H_{nex}$*. In this example, we assume* $P_{mav}(h_1) = P_{mav}(h_2) = P_{mav}(h_3) > P_{mav}(h_4)$*. We remove* h_4 *from* H_{nex} *and let* $h_{mav} = h_4$*. We assign* $T_{clk}(h_j) = 2.5\,\text{ns} \cdot 2 = 5.0\,\text{ns}$ *to each huddle* $h_j \in H_{nex}$ *and execute basic scheduling/binding. However, the scheduling result requires 8 CSs and violate CS constraint* S_{max}*.* $T_{clk}(h_j) = 2.5\,\text{ns}$ *is reassigned to each huddle* $h_j \in H_{nex}$ *and* $T_{clk}(h_{mav}) = 5.0\,\text{ns}$ *is assigned to* $h_{mav}(= h_4)$*. The basic scheduling/binding is executed and Fig. 6.8 shows the results. Since the results satisfy the CS constraint, we calculate* $Cost_s$*. Based on Fig. 6.8(a),* $|R(h_1)| = 2$, $|R(h_2)| = 2$, $|R(h_3)| = 2$, *and* $|R(h_4)| = 0$*. Therefore,* $Cost_s = 6112.5\,\text{fJ}$.

These processes are repeated and the final result of clock period assignment



(a)

	h_1	h_2	h_3	h_4
f_1	0	1	1	1
f_2	1	0	1	1
f_3	1	1	0	1
f_4	1	1	1	0

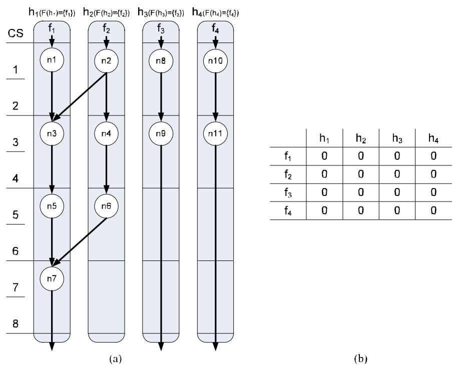
(b)

Figure 6.6: The results of initial phase. (a) The scheduled and binded DFG. (b) The data transfer table.

phase are as shown in Fig. 6.9. In this case, $|R(h_1)| = 2$, $|R(h_2)| = 2$, $|R(h_3)| = 1$, $|R(h_4)| = 1$, *and* $Cost_s$ *is calculated as follows:*

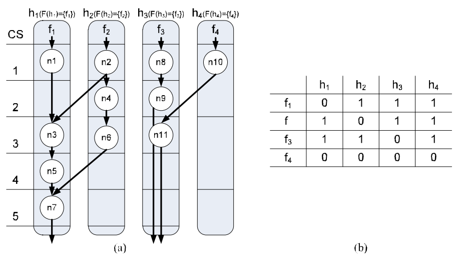
$$
\begin{aligned}
Cost_s = {} & 2 \cdot (200\,\mathrm{fJ} \cdot 5/1 + 1.5\,\mu\mathrm{W} \cdot 5 \cdot 2.5\,\mathrm{ns}) \\
& +2 \cdot (200\,\mathrm{fJ} \cdot 5/1 + 1.5\,\mu\mathrm{W} \cdot 5 \cdot 2.5\,\mathrm{ns}) \\
& +1 \cdot (200\,\mathrm{fJ} \cdot 5/2 + 1.5\,\mu\mathrm{W} \cdot 5 \cdot 2.5\,\mathrm{ns}) \\
& +1 \cdot (200\,\mathrm{fJ} \cdot 5/2 + 1.5\,\mu\mathrm{W} \cdot 5 \cdot 2.5\,\mathrm{ns}) \\
& = 4912.5\,\mathrm{fJ}.
\end{aligned}
$$

Operation scheduling/binding phase: *First,* $P_{list}(n_x)$ *is calculated for each operation node* n_x. n_{11} *whose* $P_{list}(n_{11})$ *is the minimum is picked up.* $Bind(n_{11})$ *is changed from* f_4 *to* f_1 *and basic scheduling/binding is executed. Fig. 6.10 shows the scheduling result. In this case,* $|R(h_1)| = 3$, $|R(h_2)| = 2$, $|R(h_3)| = 1$, $|R(h_4)| = 0$,


h1(F(h1)={f1}) h2(F(h2)={f2}) h3(F(h3)={f3}) h4(F(h4)={f4})
CS
f1 f2 f3 f4
n1 n2 n8 n10 n3 n4 n9 n11 n5 n6 n7
1 2 3 4 5 6 7 8
(a)
(b)

	h_1	h_2	h_3	h_4
f_1	0	0	0	0
f_2	0	0	0	0
f_3	0	0	0	0
f_4	0	0	0	0

Figure 6.7: The results of clock period assignment phase (in progress 1, CS constraint violation).


h1(F(h1)={f1}) h2(F(h2)={f2}) h3(F(h3)={f3}) h4(F(h4)={f4})
CS
f1 f2 f3 f4
n1 n2 n8 n10 n4 n9 n3 n6 n11 n5 n7
1 2 3 4 5
(a)
(b)

	h_1	h_2	h_3	h_4
f_1	0	1	1	1
f	1	0	1	1
f_3	1	1	0	1
f_4	0	0	0	0

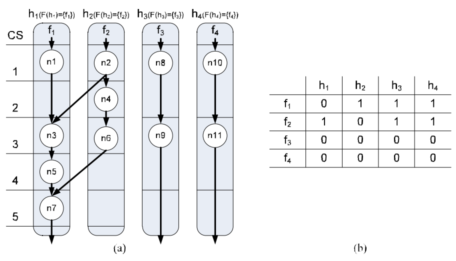
Figure 6.8: The results of clock period assignment phase (in progress 2).

and $Cost_s$ is calculated as follows:

$$\begin{aligned}
Cost_s &= 3 \cdot (200\,\mathrm{fJ} \cdot 6/1 + 1.5\,\mu\mathrm{W} \cdot 6 \cdot 2.5\,\mathrm{ns}) \\
&+2 \cdot (200\,\mathrm{fJ} \cdot 6/1 + 1.5\,\mu\mathrm{W} \cdot 6 \cdot 2.5\,\mathrm{ns}) \\
&+1 \cdot (200\,\mathrm{fJ} \cdot 6/2 + 1.5\,\mu\mathrm{W} \cdot 6 \cdot 2.5\,\mathrm{ns}) \\
&+0 \cdot (200\,\mathrm{fJ} \cdot 6/2 + 1.5\,\mu\mathrm{W} \cdot 6 \cdot 2.5\,\mathrm{ns}) \\
&= 6735\,\mathrm{fJ} \\
&> MinCost (= 4912.5\,\mathrm{fJ}).
\end{aligned}$$

Similarly, $Bind(n_{11})$ is changed to f_2 and we calculate $Cost_s = 6735$. $Bind(n_{11})$ is changed to f_3 and we calculate $Cost_s = 6135$. Finally, the Cost when $Bind(n_{11}) = f_4$ is the minimum and the binding about n_{11} is not changed.

These processes are repeatedly executed to each node and Fig. 6.11 shows the results of operation scheduling/binding phase. Fig. 6.11(a) and (b) are also the outputs of scheduling/binding. □


h1(F(h1)={f1}) h2(F(h2)={f2}) h3(F(h3)={f3}) h4(F(h4)={f4})
CS
1
2
3
4
5
f1 f2 f3 f4
n1 n2 n3 n4 n5 n6 n7 n8 n9 n10 n11
(a)
(b)

	h_1	h_2	h_3	h_4
f_1	0	1	1	1
f_2	1	0	1	1
f_3	0	0	0	0
f_4	0	0	0	0

Figure 6.9: The final results of clock period assignment phase.

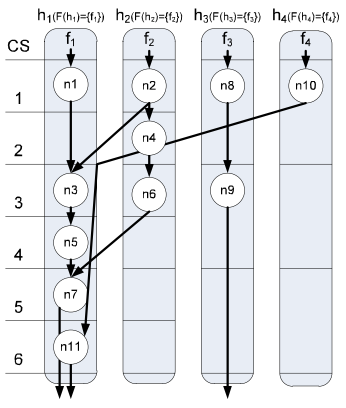
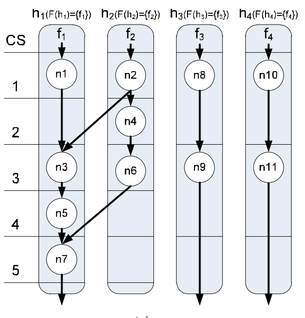

h1(F(h1)={f1}) h2(F(h2)={f2}) h3(F(h3)={f3}) h4(F(h4)={f4})
CS
1
2
3
4
5
6
f1 f2 f3 f4
n1 n2 n8 n10 n4 n3 n6 n9 n5 n7 n11

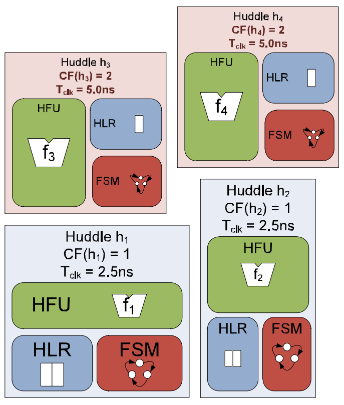
Figure 6.10: The result of operation scheduling/binding phase in progress.

Table 6.4: $CP(n_x, f_j)$ for each operation node n_x and each FU f_j.

	f_1	f_2	f_3	f_4
n_1	**4**	4	6	6
n_2	**4**	4	6	6
n_3	**3**	3	5	5
n_4	**3**	3	5	5
n_5	**2**	2	4	4
n_6	**2**	2	4	4
n_7	**1**	1	2	2
n_8	**2**	2	4	4
n_9	**1**	1	2	2
n_{10}	**2**	2	4	4
n_{11}	**1**	1	2	2



(a)



(b)

Figure 6.11: The result of operation scheduling/binding phase and the output of the scheduling/binding step.

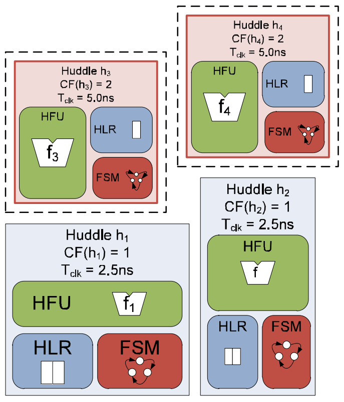


Figure 6.12: The result of register/controller synthesis.

6.4.3 Register/controller synthesis

In the register/controller synthesis step, register and controller configuration in each huddle is determined according to the result of a scheduling/binding step. The same algorithm as HDR architectures proposed in Chapter 3 is applied to register and controller configurations in each huddle.

Example 6.4. *In register/controller synthesis, the register and controller configuration in each huddle is determined according to the result of scheduling/binding step. The input and the output are as follows:*

Input *:*

- *scheduled CDFG (Fig. 6.11(a))*

Output *:*

- *huddle configuration (Fig. 6.12).*

In the iteration process, we estimate each huddle area based on virtual area. In Fig. 6.12, the areas of huddle h_3 and h_4 in this phase are smaller than those in the initial process. The virtual areas of huddle h_3 and h_4 are calculated as in Section 6.4. Therefore, the virtual areas of huddle h_3 and h_4 do not decrease in Fig. 6.12. □

6.4.4 Unhuddling

In unhuddling, we divide huddles which violate the huddle size constraint. In the iteration process, virtual area of each huddle is used. Therefore there are some

Algorithm 6.5 Unhuddling

```
1: for all $h_j$ do
2:    if $h_j$ violate the huddle size constraint then
3:       for $f_i \in F(h_j)$ do
4:          if there are any functional units inside $h_j$ other than $f_i$ then
5:             huddle $h_{vacant}$.
6:             for all $h_k$ do
7:                if $h_k$ is empty then
8:                   $h_{vacant} \leftarrow h_k$, goto step 11.
9:                end if
10:            end for
11:            $f_i$ is assigned to $h_{vacant}$.
12:            $A_{virtual}(h_{vacant}) \leftarrow A_{real}(h_{vacant})$.
13:         else
14:            $A_{virtual}(h_j) \leftarrow A_{real}(h_j)$.
15:         end if
16:      end for
17:   end if
18: end for
```

huddles to which no huddles are bound. We call the huddles *empty huddles*. If huddle h_j violate the huddle size constraint, it contains too many functional units. We pick up a functional unit f_i inside h_j and assign it to a empty huddle h_{vacant}. Virtual area of h_{vacant} is assumed to be the real area of h_{vacant}. We repeat the process until only one functional unit is assigned to h_j. Finally, we recalculate the virtual area of h_j based on the real area of h_j. Algorithm 6.5 shows the algorithm of unhuddling step. In (viii) floorplanning-directed huddling step, the useless space and overlapping of huddles will be resolved.

Example 6.5. *The input and the output are as follows:*

Input *:*

- *huddle configuration (Fig. 6.13(a))*

Output *:*

- *huddle configuration (Fig. 6.13(b)).*

As shown in Fig. 6.13(a), huddle h_3 contains two FUs f_3 and f_4. We assume that h_3 violates the huddle size constraint. First, we pick up huddle h_4, which is an empty huddle, and let $h_{vacant} = h_4$. We transfer f_3, associated registers (HLRs), and controllers (FSM) from h_3 to h_4. The virtual area of h_4 is calculated. Finally, we calculated the virtual area of h_3 and unhuddling step is finished. □

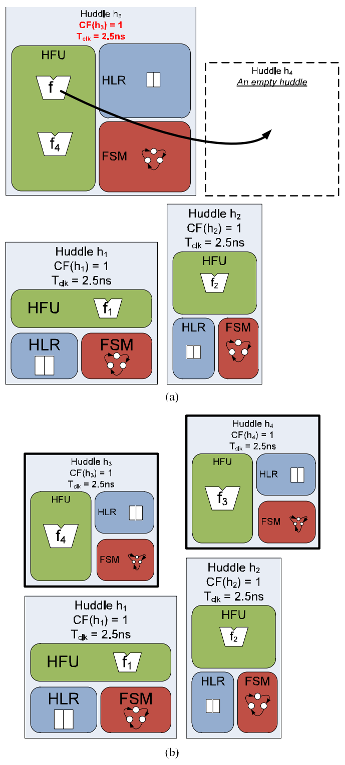


Figure 6.13: The result of unhuddling. (a) The input huddle configuration. (b) The output huddle configuration.

6.4.5 Floorplanning and floorplanning-directed huddling

In floorplaning-directed huddling, huddle placement as well as its height and width is optimized by using a simulated annealing (SA) strategy based on a sequence-pair representation [28]. In this step, we consider the four moves as follows:

Move 1: Select two elements and exchange them in Γ_+.

Move 2: Select two elements and exchange them in Γ_+ and Γ_-.

Move 3: Select one element and change its aspect ratio.

Move 4: Select functional unit f_i and transfer it from the huddle h_j to the huddle $h_k (\neq h_j)$.

In floorplanning, we only consider Move 1, Move 2, and Move 3. In SA optimization, its cost function *cost* is expressed by

$$cost = \alpha \frac{V \cdot S_{max}}{T_{clkmin}} + \beta \frac{E_{PlaceDep}}{E_{PlaceIndep}} \tag{6.16}$$

where T_{clkmin} is the minimum clock period constraint, V is the sum of violations, S_{max} is the control step constraint, $E_{PlaceDep}$ is the placement dependent energy consumption which includes wire dynamic energy consumption and clock tree energy consumption, and $E_{PlaceIndep}$ is the placement independent energy consumption which includes energy consumption of functional units, registers, and controllers. α and β are parameters.

The initial solution of floorplanning and floorplanning-directed huddling at each iteration is the solution represented by its sequence-pair of the previous result so that the entire iteration in Fig. 6.3 can converge gradually. Initial temperature T_i in floorplanning and floorplanning-directed huddling at the i-th iteration of the synthesis flow is computed by

$$T_{i+1} = KT_i \tag{6.17}$$

where K is also a parameter and set to be $K < 1$.[1]

Example 6.6. *In floorplanning directed huddling, huddle placement and huddle configuration are optimized simultaneously. The input and the output are as follows:*

Input *:*

- *huddle configuration (Fig. 6.12).*

Output *:*

- *huddle configuration (Fig. 6.14).*

In the example, functional unit f_4 transfers into huddle h_3. □

6.4.6 Virtual area adaptation

Virtual area may increase interconnection delay between huddles as the iterations proceed. To solve this problem, we should gradually decrease the difference between virtual area and real area.

We execute *virtual area adaptation* after floorplanning-directed huddling. Because this step is just before scheduling/binding at the next iteration, we can use virtual area closer to real area at the next iteration.

Virtual area adjustment is executed as follows:

1. Let $A_{dif}(h_j) = A_{virtual}(h_j) - A_{real}(h_j)$ be the difference between real area $A_{real}(h_j)$ and virtual area $A_{virtual}(h_j)$ of the huddle h_j.

[1] In the experiments, $\alpha = 100$, $\beta = 1$, and $K = 0.9$ were set.

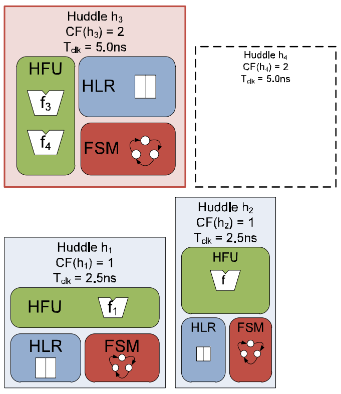


Figure 6.14: The result of floorplanning directed huddling.

2. We set $A_{virtual}(h_j) = A_{real}(h_j) + \phi \cdot A_{dif}(h_j)$,

where ϕ is an adaptive parameter. In order to decrease ϕ as the iterations proceed, we set $\phi = \max\{1 - 0.05i, 0\}$ at the i-th iteration. Finally of the virtual area adjustment, the useless space is eliminated.

Example 6.7. *In virtual area adaptation, we decrease the virtual area. The input and the output are as follows:*

Input :

- *huddle configuration (Fig. 6.15(a)).*

Output :

- *huddle configuration (Fig. 6.15(b)).*

In the example, the virtual area of huddle h_3 and h_4 is decreased. □

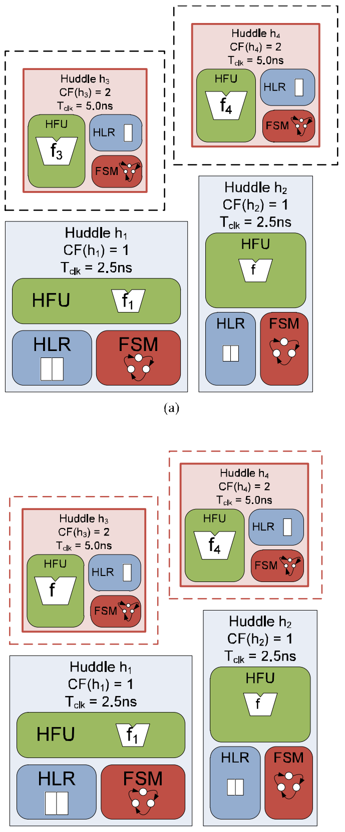


Figure 6.15: The result of virtual area adaptation. (a) The input huddle configuration. (b) The output huddle configuration.

Table 6.5: Components information.

	Area [μm^2]	Delay [ns]	Dynamic energy [fJ]	Leak power [μW]
Adder	386	1.22	64.0	3.20
Multiplier	2161	2.70	788.0	16.50
Subtractor	417	1.27	67.4	3.50
Devider	6066	10.21	1601.1	69.80
Shifter	294	0.89	52.1	2.10
Comparator	116	0.83	11.8	0.67
16bit AND	68	0.66	3.6	0.66
Memory access	–	2.70	–	–
16bit MUX	36	0.21	6.3	0.56
16bit Register	309	0.45	194.0	1.35

6.5 Experimental Results

The proposed algorithm has been implemented in C++ on UNIX 2.5 GHz $\times 2$ with 16 GB memory. The algorithm has been applied to DCT (a discrete cosine transform algorithm for 8×8 pixels, 48 nodes), EWF3 (three elliptic wave filters are serially connected, 102 nodes), FIR filter (a seventh order finite impulse response filter, 75 nodes), JACOBI (Jacobi method to solve linear equations with four unknown variables, 48 nodes), PARKER [29] (22 nodes, including conditional branches), and COPY (provided by a company, 378 nodes, including conditional branches). Table 6.5 shows the components specifications. Memories were assumed to be prepared outside. Therefore the memory access was assumed to be a special type of functional unit as shown in Table 6.5. All the functional units were assumed to have a bit width of 16, and their specifications were obtained by synthesizing them beforehand based on the CMOS 90 nm technology. The minimum clock period constraint was given to be 2.5 ns in all experiments. Controllers were synthesized by Synopsys Design Compiler in each iteration. The interconnection delays were assumed to be a proportion to square of the wiring length and an interconnection delay is set to be 1 ns when wiring length is 250μm. A clock tree was considered for all the huddles with the same clock frequencies and its energy is obtained by using the equations in [43].

Table 6.6, Table 6.7, and Table 6.8 summarize the experimental results. The proposed algorithm (*HDR-mcd* in Table 6.6 and Table 6.7) has been compared to MCAS for RDR architectures [8] (*RDR* in Table 6.6 and Table 6.7), MH^4 for HDR

Table 6.6: Experimental results other than energy consumption

App. (S_{max})	FUs	Method	Steps	$\#h$	Area [μm^2]	CPU time [s]
DCT	ADD×4	RDR [8]	12	9	129600	198.5
(12)	MUL×4	MH^4 (Single)	12	6	59527	879.1
		CGHDR [1]	11	5	62468	577.4
		HDR-mcd	12	7	55233	545.3
EWF3	ADD×4	RDR [8]	59	9	176400	207.0
(61)	MUL×4	MH^4 (Single)	60	5	47196	749.5
		CGHDR [1]	89*	3	68850	410.5
		HDR-mcd	60	6	49486	457.0
FIR	ADD×4	RDR [8]	29	6	86400	125.6
(31)	MUL×4	MH^4 (Single)	31	4	31395	567.0
		CGHDR [1]	31	4	44688	533.7
		HDR-mcd	30	5	50525	511.1
JACOBI	ADD×2	RDR [8]	31	4	57600	76.3
(31)	SUB×1	MH^4 (Single)	31	6	33200	376.5
	MUL×2	CGHDR [1]	31	5	31191	529.8
	DIV×2	HDR-mcd	31	6	33920	433.7
COPY	ADD×3, SUB×1	RDR [8]	181*	16	2250000	751.6
(172)	Comp×1, Rshift×2	MH^4 (Single)	171	9	283544	2106.1
	AND×1, MUL×5	CGHDR [1]	179*	11	541233	4174.1
		HDR-mcd	172	12	325360	6691.9
PARKER	ADD×2	RDR [8]	7	4	57600	97.0
(7)	SUB×2	MH^4 (Single)	7	2	11661	290.9
	Comp×1	CGHDR [1]	7	1	10000	197.1
		HDR-mcd	6	5	11110	339.6

* In which the S_{max} constraint is violated.

architecture synthesis algorithm with single supply voltage (*MH*^{4} *(Single)* in Table 6.6 and Table 6.7), and an HDR architecture synthesis algorithm with clock gaiting [1] (*CGHDR* in Table 6.6 and Table 6.7). All the three conventional methods consider floorplannning during the synthesis, but they only consider single clock domain. In Table 6.6 and Table 6.7, "S_{max}" shows the CS constraint S_{max}. "CPU time" shows CPU time to synthesize each circuit. "Dynamic", "Leak", "Wire", and "Clock Tree" represent dynamic energy consumption, leakage energy consumption, wire dynamic energy consumption, and clock tree energy consumption. "All" shows the sum of "Dynamic", "Leak", "Wire", and "Clock tree". "$\#h$" in Table 6.6 shows the number of islands or huddles of RDR, HDR, CGHDR, and HDR-mcd.

The experimental results show that all energy consumption of HDR-mcd is

Table 6.7: Energy comparison of experimental results.

App. (S_{max})	Method	Dynamic [pJ]	Leak [pJ]	Wire [pJ]	Clock Tree [pJ]	All [pJ]
DCT	RDR [8]	98.4	19.4	38.4	149.3	305.7
(12)	HDR	98.3	16.7	34.3	108.3	257.5
	CGHDR [1]	73.4	15.4	35.5	69.9	194.2
	HDR-mcd	73.7	15.2	26.2	55.5	**170.7**
EWF3	RDR [8]	280.6	102.2	103.9	600.6	1087.3
(61)	HDR	277.2	76.6	85.6	300.2	739.5
	CGHDR [1]	326.6	166.7	111.3	477.5	1082.1
	HDR-mcd	217.7	76.9	77.6	235.6	**607.8**
FIR	RDR [8]	128.6	28.3	53.0	203.4	413.3
(31)	HDR	118.0	25.7	41.0	136.0	320.7
	CGHDR [1]	108.5	38.3	48.9	119.1	314.9
	HDR-mcd	84.7	33.9	46.9	65.0	**230.5**
JACOBI	RDR [8]	78.9	118.8	21.2	136.2	355.2
(31)	HDR	101.4	122.7	18.7	103.6	346.4
	CGHDR [1]	62.1	118.2	17.8	63.3	**261.4**
	HDR-mcd	82.2	122.0	12.6	83.5	300.3
COPY	RDR [8]	5334.8	1655.0	1421.6	17175.3	25586.7
(172)	HDR	3763.9	1215.1	564.1	6754.6	12297.7
	CGHDR [1]	4134.2	1749.1	664.7	6899.7	13447.7
	HDR-mcd	1599.1	1298.1	615.6	2476.4	**5989.2**
PARKER	RDR [8]	17.3	1.6	19.7	35.3	74.0
(7)	HDR	14.5	1.7	20.8	17.8	54.8
	CGHDR [1]	9.9	1.0	20.5	13.2	44.6
	HDR-mcd	14.8	1.0	8.4	14.1	**38.2**

Table 6.8: The results of clock assignment about HDR-mcd.

App.	CF			
	1(= 2.5 ns)	2	4	8
DCT	3	4		
EWF3	2	4		
FIR		5		
JACOBI	4	1	1	
COPY	1	6	3	2
PARKER	4	1		

reduced by a maximum of 76.6% and an average of 32.5% compared with the other algorithms. The dynamic energy consumption of functional units, registers, and controllers is also reduced by a maximum of 70.0% and an average of 19.2% compared with the other algorithms. Most of the dynamic energy is the energy consumption of registers. HDR-mcd can reduce the dynamic energy because multiple clock domains can reduce the switching energy of registers. The clock tree energy of HDR-mcd is reduced by a maximum of 85.6% and an average of 41.3% compared with the other algorithms. Because a clock tree is required for each of clock, multiple clock domains may increase the clock tree energy consumption. However the HDR-mcd can reduce the energy consumption more than the energy overhead by increasing of the number of clock trees. The wire dynamic energy of HDR-mcd is reduced by maximum of 59.7% and an average of 26.8% compared with the other algorithms. This is because the cost function of the floorplanning algorithm (Eqn. (6.16)) consider the energy depending on placement such as wire energy, but the other algorithms do not consider. The leakage energy of HDR-mcd is reduced by an average of 10.9% compared with the other algorithms. To achieve further reduction of the leakage energy is the future work.

6.6 The Combination of MSV and MCD

For further comparison, MSV and MCD are also applied simultaneously. First, I describe the target architecture which is an extended HDR-mcd and AVHDR in Chapter 5. Next, the HLS algorithm for the architecture is described. Finally, the experimental results are described.

HDR-mcv architecture

I propose HDR-mcv architecture which is the combination of HDR-mcd architecture proposed in this Chapter and AVHDR architecture proposed in Chapter 5.

In HDR-mcv, MSV and MCD are considered but PG and/or DMSV are not considered. The combination of the all low-energy LSI design techniques is the future work. In each HDR-mcv huddle, two types of power supply rails are prepared for applying MSV like AVHDR architecture. One is for *functional logic (FL)* and the other is for *synchronized logic (SL)*. Clock period is assigned to each huddle.respectively like HDR-mcd architecture. An HDR-mcv example is shown in Fig. 6.16, in which a huddle, h, consists of FL and SL:

Functional Logic (FL)

FL is composed of several HFUs, and the voltage is assigned to each HFU respectively.

Huddled Functional Unit (HFU): A dedicated functional unit and I/O level converters in h. Each pair of a functional unit and I/O level converters is connected to its dedicated power supply rail. In HDR-mcd, Its supply voltages are not changed in runtime and PG cannot be applied.

Synchronized Logic (SL)

SL is composed of HLRs, FSM, and HLCs, and only one constant voltage is assigned to FVL.

Huddled Local Registers (HLRs): Dedicated local registers in h and input multiplexers. HFUs can only access the HLRs in h. We ignore the interconnection dela ly close to the HLRs.

Finite State Machine (FSM): A dedicated controller in h. FSM controls the AVFUs and the HLRs in h.

Huddled Level Converters (HLCs): Dedicated level converters in h. HLCs are used during inter-huddle data transfer from lower voltage huddles.

The HLS Algorithm for HDR-mcv architecture

A new HLS algorithm for HDR-mcv architecture is proposed. A virtual-area-based iterative refinement flow proposed in Chapter 4 is used in the algorithm. The proposed algorithm is composed of the following eight steps in each iteration:

- **initial huddling,**
- **scheduling/binding,**
- **register/controller synthesis,**

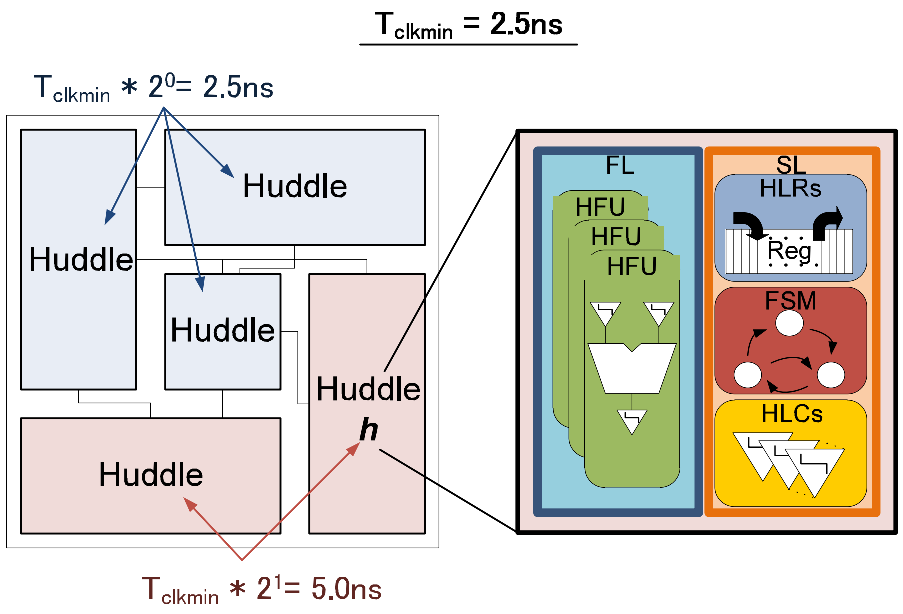


Figure 6.16: An HDR-mcv architecture.

- **huddle voltage adaptation,**
- **unhuddling,**
- **floorplanning,**
- **floorplanning-directed huddling,**
- **virtual area adaptation.**

Initial huddling, register/controller synthesis, unhuddling, floorplanning, floorplanning-directed huddling, and virtual area adaptation are proposed in previous Sections. Huddle voltage adaptation is proposed in Section 5.4.4. For further details about the steps, refer to the respective Sections. In the rest of this section, I will describe the proposed scheduling/binding in details. Fig. 6.17 shows the algorithm.

Scheduling/binding

The scheduling/binding problem here is, for given a CDFG $G(N, E)$, a minimum clock period constraint T_{clkmin}, a control step constraint S_{max}, a set of functional units F, and huddle configuration, to find scheduling and functional unit binding of every node in a given CDFG, to determine clock periods $T_{clk}(h_j)$ of each huddle h_j, and to determine supply voltages assigned (v_l, v_m, and v_h) to given FL and

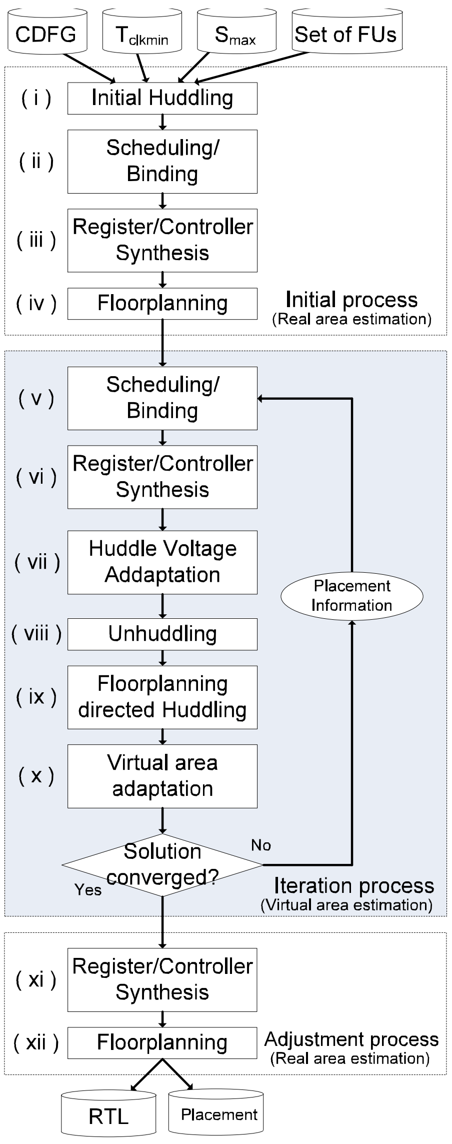

CDFG
T_{clkmin}
S_{max}
Set of FUs
(i) Initial Huddling
(ii) Scheduling/ Binding
(iii) Register/Controller Synthesis
(iv) Floorplanning
Initial process (Real area estimation)
(v) Scheduling/ Binding
(vi) Register/Controller Synthesis
(vii) Huddle Voltage Addaptation
(viii) Unhuddling
(ix) Floorplanning directed Huddling
(x) Virtual area adaptation
Placement Information
Solution converged?
No
Yes
Iteration process (Virtual area estimation)
(xi) Register/Controller Synthesis
(xii) Floorplanning
Adjustment process (Real area estimation)
RTL
Placement

Figure 6.17: Proposed HLS algorithm for HDR-mcv architecture.

SL, so as to minimize the total energy consumption while meeting the control step constraint.

The scheduling/binding is composed of the five phases: initial phase, SL voltage assignment phase, FL voltage assignment phase, clock period assignment phase, operation scheduling/binding phase.

(1) Initial phase:
In the initial phase, scheduling and binding are executed according to huddle placement and clock periods obtained by the previous iteration.

(2) SL voltage assignment phase:
In the SL voltage assignment phase, we search the voltage assignment so as to minimize total energy consumption. The SL voltage assignment phase is the same algorithm as the FVL voltage decreasing phase proposed in Section 5.4.2.

(3) FL voltage assignment phase:
In the FL voltage assignment phase, we search the voltage assignment so as to minimize total energy consumption. The FL voltage assignment phase is the same algorithm as the AVL voltage decreasing phase proposed in Section 5.4.2.

(4) Clock period assignment phase:
In the clock period assignment phase, we search the clock periods assignment so as to minimize total energy consumption. The clock period assignment phase is the same algorithm as the clock period assignment phase proposed in Section 6.4.2.

(5) Operation scheduling/binding phase:
In the operation scheduling/binding phase, operations are assigned to control steps and FUs so as to minimize total energy consumption. The operation scheduling/binding phase is the same algorithm as the operation scheduling/binding phase proposed in Section 6.4.2.

For further details, refer to the respective Sections.

Fig. 6.18 shows the scheduling/binding algorithm. The order of the phases is determined according to the ratio of energy consumption. When the leak energy is bigger than the clock energy in previous iteration, we first assign voltage to each SL. Next, we assign voltage to each FL. Finally, we assign clock period to each huddle. If low voltages are assigned to the SLs and FLs, we can reduce not only the dynamic energy consumption but also the leak energy consumption. If long clock periods are assigned to huddles, the leak energy cannot be reduced. On the other hand, when the clock energy is bigger than the leak energy in previous iteration, we first assign voltage to each SL. Next, we assign clock period to each huddle.

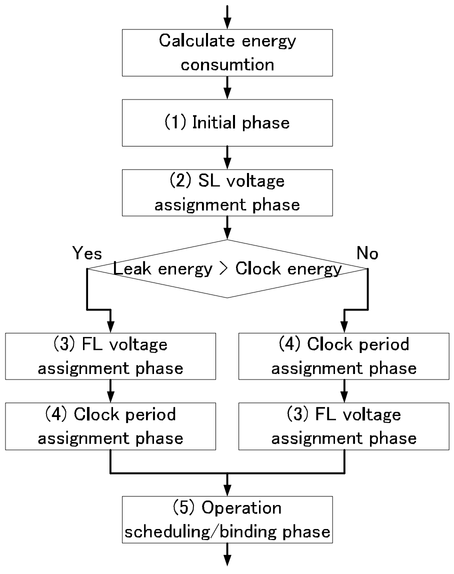

Calculate energy consumtion
(1) Initial phase
(2) SL voltage assignment phase
Yes
Leak energy > Clock energy
No
(3) FL voltage assignment phase
(4) Clock period assignment phase
(4) Clock period assignment phase
(3) FL voltage assignment phase
(5) Operation scheduling/binding phase

Figure 6.18: The scheduling/binding algorithm.

Finally, we assign voltage to each FL. If low voltages are assigned to the SLs and long clock periods are assigned huddles, we can reduce not only the dynamic energy consumption but also the clock energy consumption. If low voltages are assigned to FLs, the clock energy cannot be reduced. In order to reduce the energy which accounts for a large percentage, we assign voltages and clock period in the order.

We design a cost function $Cost_{mcv}$. $Cost_{mcv}$ is the sum of the energy consumption of functional units and registers and is calculated as:

$$\begin{aligned} Cost_s = & \sum_{\forall n_x} E(Bind(n_x)) + \sum_{\forall f_i} P_l(f_i) S_{max} T_{clkmin} + \\ & \sum_{\forall h_j} \left| R(h_j) \right| \cdot \left\{ R_d S_{result} / CF(h_j) + R_l S_{result} T_{clkmin} \right\} \end{aligned} \tag{6.18}$$

where $P_l(f_i)$ is the leak power of functional unit f_i, $R(h_j)$ is the set of registers in h_j, R_d is a dynamic energy consumption of a register, R_l is a leak power of a register, and S_{result} is the end CS of the scheduling result ($S_{result} \leq S_{max}$). $R(h_j)$ of each huddle is estimated by left-edge based register binding algorithm. In all phases, we select the voltage assignment and the clock assignment which have the minimum value of $Cost_{mcv}$.

Experimental results

The proposed algorithm has been implemented in C++ on UNIX 2.5 GHz $\times 2$ with 16 GB memory. The algorithm has been applied to DCT (a discrete cosine transform algorithm for 8×8 pixels, 48 nodes), EWF3 (three elliptic wave filters are serially connected, 102 nodes), FIR filter (a seventh order finite impulse response filter, 75 nodes), JACOBI (Jacobi method to solve linear equations with four unknown variables, 48 nodes), PARKER [29] (22 nodes, including conditional branches), and COPY (provided by a company, 378 nodes, including conditional branches). Table 6.9 shows the components specification. Table 6.10 shows the level converters specification. Memories were assumed to be prepared outside. Therefore the memory access was assumed to be a special type of functional unit as shown in Table 6.9. All the functional units were assumed to have a bit width of 16, and their specifications were obtained by synthesizing them beforehand based on the CMOS 90 nm technology. Selectable voltages were assumed to be as $v_l = 0.8\,\mathrm{V}$, $v_m = 1.0\,\mathrm{V}$, and $v_h = 1.2\,\mathrm{V}$. The minimum clock period constraint was given to be 1.5 ns in all experiments. Controllers were synthesized by Synopsys Design Compiler in each iteration. The interconnection delays were assumed to be a proportion to square of the wiring length and an interconnection delay is set to be 1 ns when wiring length is 250μm. A clock tree was considered for all the huddles with the same clock frequencies and its energy is obtained by using the equations in [43].

Table 6.11, Table 6.12, and Table 6.13 summarize the experimental results. The proposed algorithm (*HDR-mcv* in Table 6.11 and Table 6.12) has been compared to a traditional shared-register architecture synthesis algorithm (*SR* in Table 6.11 and Table 6.12), MCAS for RDR architectures [8] (*RDR* in Table 6.11 and Table 6.12), MH^4 for HDR architecture synthesis algorithm with single supply voltage (*MH^4 (Single)* in Table 6.11 and Table 6.12), SAMCID for HDR-mcd architecture synthesis algorithm with MCD (*HDR-mcd* in Table 6.11 and Table 6.12), MH^4 for HDR architecture synthesis algorithm with MSV (*MH^4* in Table 6.11 and Table 6.12), and HDR-mcv architecture synthesis algorithm with only MSV (*HDR-mcv (MSV)* in Table 6.11 and Table 6.12). In JACOBI, the proposed algorithm has been further compared to SAAV for AVHDR architecture synthesis algorithm with DMSV (*SAAV* in Table 6.11 and Table 6.12). In Table 6.6 and Table 6.7, "S_{max}" shows the CS constraint S_{max}. "*#h.*" shows shows the number of huddles of MH^4 (Single), HDR-mcd, MH^4, HDR-mcv (MSV) and HDR-mcv.. "Tech." shows applied energy-efficient LSI design techniques. "CPU time" shows CPU time to synthesize each circuit. "Dynamic", "Leak", "Wire", and "Clock Tree" represent dynamic energy consumption, leakage energy consumption, wire dynamic energy consumption, and clock tree energy consumption. "All" shows the sum of "Dy-

Table 6.9: Informations of components.

Adder	Area :	386	μm^2
	1.2V	1.0V	0.8V
Delay[ns]	0.75	1.22	2.71
Dynamic energy[fJ]	103.97	64	33.22
Leak power[μW]	5.97	3.2	1.74
Subtractor	Area :	417	μm^2
	1.2V	1.0V	0.8V
Delay[ns]	0.78	1.27	2.82
Dynamic energy[fJ]	109.49	67.4	34.99
Leak power[μW]	6.53	3.5	1.9
Multiplier	Area :	2161	μm^2
	1.2V	1.0V	0.8V
Delay[ns]	1.65	2.7	6
Dynamic energy[fJ]	1324.38	788	495.13
Leak power[μW]	29.7	16.5	8.25
Comparator	Area :	116	μm^2
	1.2V	1.0V	0.8V
Delay[ns]	0.51	0.83	1.84
Dynamic energy[fJ]	19.17	11.8	6.13
Leak power[μW]	1.25	0.67	0.36
16bit Register	Area :	330	μm^2
	1.2V	1.0V	0.8V
Delay[ns]	0.29	0.47	1.05
Dynamic energy[fJ]	305.22	187.88	97.53
Leak power[μW]	2.75	1.47	0.8
16bit MUX	Area :	576	μm^2
	1.2V	1.0V	0.8V
Delay[ns]	0.13	0.21	0.47
Dynamic energy[fJ]	162.72	100.16	52
Leak power[μW]	16.65	8.93	4.86

Table 6.10: Informations of level converters.

V_{in}–V_{out}	Area [μm^2]	Delay [ns]	Dynamic [fJ]	Leak [μW]
1.0V–1.2V	113	0.042	1.92	2.47
0.8V–1.2V	113	0.074	3.26	2.12
0.8V–1.0V	113	0.059	2.22	3.55

namic”, “Leak”, “Wire”, and “Clock tree”. “*#CT*” shows the number of clock trees.

The experimental results show that all energy consumption of HDR-mcv is reduced by a maximum of 85.4% and an average of 57.0% compared with the other algorithms. The dynamic energy consumption of functional units, registers, and controllers is also reduced by a maximum of 87.5% and an average of 55.4% compared with the other algorithms. Most of the dynamic energy is the energy consumption of registers. HDR-mcv can reduce the dynamic energy because MSV and MCD can reduce the switching energy of registers. The clock tree energy of HDR-mcv is reduced by a maximum of 89.0% and an average of 65.2% compared with the other algorithms. The wire dynamic energy of HDR-mcv is reduced by maximum of 89.1% and an average of 48.5% compared with the other algorithms. The leakage energy of HDR-mcv is reduced by an average of 30.3% compared with the other algorithms. However, SAAV reduces the leakage energy more than HDR-mcv because SAAV can cut off the leakage energy through PG and/or DMSV. To achieve further reduction of the leakage energy utilizing with PG and/or DMSV is the future work.

The effects of MSV and MCD are compared. Table 6.14 shows the energy comparison. The proposed HLS considering MCD achieves 26.1% energy-saving. The proposed HLS considering MSV achieves 52.9% energy-saving. In the architecture and HLS, MSV can reduce energy more than MCD. On the other hand, the proposed HLS considering MCD and MSV simultaneously achieves 65.8% energy-saving. The energy reduction effects of MCD and MSV are orthogonal as follows:

$$0.739 \times 0.471 = 0.348069 \simeq 0.342.$$

Therefore, in the proposed HDR-mcv, MCD and MSV can reduce energy consumption almost independently.

6.7 Conclusion

In this chapter, I propose *an HDR-mcd architecture*, which integrates periodically all-in-phase based multiple clock domains and multi-cycle interconnect communication into HLS. Next, I propose a high-level synthesis algorithm for *HDR-mcd*. Experimental results show that the proposed algorithm achieves 32.5% energy-saving compared with conventional algorithms. Furthermore, the proposed method which can apply MCD and MSV simultaneously achieves 57.0% energy-saving compared with the conventional methods.

Table 6.11: Experimental results other than energy consumption.

App. (S_{max})	FUs	Method	Tech.	Steps	$\#h$	Area [μm^2]	CPU time [s]
DCT	ADD×4	SR	–	14	–	51529	25.7
(15)	MUL×4	RDR	–	13	–	96800	191.4
		MH^4 (Single)	–	12	8	56896	1003.2
		HDR-mcd	MCD	12	8	56595	812.1
		MH^4	MSV	15	8	68944	826.1
		HDR-mcv (MSV)	MSV	15	8	68523	820.7
		HDR-mcv	MSV + MCD	14	7	58860	760.6
EWF3	ADD×4	SR	–	89*	–	61009	29.0
(65)	MUL×4	RDR	–	72*	–	156800	159.1
		MH^4 (Single)	–	64	8	48970	568.7
		HDR-mcd	MCD	59	6	62510	923.5
		MH^4	MSV	62	8	103788	501.6
		HDR-mcv (MSV)	MSV	63	5	53352	443.0
		HDR-mcv	MSV + MCD	64	7	65856	475.1
FIR	ADD×4	SR	–	41	–	65025	26.8
(50)	MUL×4	RDR	–	43	–	99225	228.0
		MH^4 (Single)	–	45	8	25066	613.3
		HDR-mcd	MCD	45	8	38080	642.7
		MH^4	MSV	48	9	57568	665.0
		HDR-mcv (MSV)	MSV	48	8	54936	850.4
		HDR-mcv	MSV + MCD	45	5	40803	554.8
JACOBI	ADD×2	SR	–	35	–	36864	25.9
(35)	SUB×1	RDR	–	34	–	60000	52.9
	MUL×2	MH^4 (Single)	–	31	6	33565	579.1
	DIV×2	HDR-mcd	MCD	34	7	44600	428.8
		MH^4	MSV	35	7	53862	433.7
		HDR-mcv (MSV)	MSV	35	6	42570	596.4
		SAAV	DMSV	35	5	32292	632.3
		HDR-mcv	MSV + MCD	31	5	34352	552.6
COPY	ADD×3	SR	–	272*	–	331776	50.8
(250)	SUB×1	RDR	–	266*	–	1000000	657.6
	Comp×1	MH^4 (Single)	–	236	14	402560	3617.9
	Rshift×2	HDR-mcd	MCD	248	14	488250	12608.9
	AND×1	MH^4	MSV	224	14	441030	3903.4
	MUL×5	HDR-mcv (MSV)	MSV	231	14	389270	2642.0
		HDR-mcv	MSV + MCD	248	14	458414	19094.0
PARKER	ADD×2	SR	–	7	–	10201	25.0
(10)	SUB×2	RDR	–	7	–	20000	119.1
	Comp×1	MH^4 (Single)	–	9	5	12190	951.8
		HDR-mcd	MCD	7	5	10168	796.5
		MH^4	MSV	10	5	23856	296.1
		HDR-mcv (MSV)	MSV	10	3	10192	235.3
		HDR-mcv	MSV + MCD	6	3	11644	304.5

* In which the S_{max} constraint is violated.

Table 6.12: Energy comparison of experimental results.

App. (S_{max})	Method	Dynamic [pJ]	Leak [pJ]	Wire [pJ]	Clock Tree [pJ]	All [pJ]	%
DCT (15)	SR	109.5	16.3	92.9	98.9	317.6	100.0
	RDR	171.7	21.2	60.0	183.0	435.9	137.3
	MH^4 (Single)	149.1	16.8	41.0	115.1	322.0	101.4
	HDR-mcd	146.1	18.4	39.7	107.6	311.8	98.2
	MH^4	127.1	18.5	34.0	126.4	306.1	96.4
	HDR-mcv (MSV)	69.0	9.5	20.0	67.1	165.6	52.1
	HDR-mcv	54.5	9.0	18.1	33.3	114.9	36.2
EWF3 (65)	SR	392.0	142.3	258.7	513.7	1306.7	100.0
	RDR	493.3	122.0	158.6	788.7	1562.6	119.6
	MH^4 (Single)	449.9	90.0	116.2	473.0	1129.1	86.4
	HDR-mcd	336.5	85.8	107.5	335.8	865.7	66.3
	MH^4	410.9	85.2	111.3	441.6	1049.0	80.3
	HDR-mcv (MSV)	159.0	37.5	53.1	187.3	436.9	33.4
	HDR-mcv	141.8	44.2	49.7	135.7	371.4	28.4
FIR (50)	SR	313.9	85.9	151.3	353.7	904.8	100.0
	RDR	260.4	32.3	76.9	382.0	751.6	83.1
	MH^4 (Single)	258.3	30.1	45.1	242.8	576.3	63.7
	HDR-mcd	184.2	43.2	40.4	145.2	413.0	45.6
	MH^4	156.4	22.5	28.0	217.3	424.1	46.9
	HDR-mcv (MSV)	125.3	20.4	21.7	153.6	321.0	35.5
	HDR-mcv	103.3	27.3	16.5	69.0	216.1	23.9
JACOBI (35)	SR	134.6	110.5	54.5	134.0	433.7	100.0
	RDR	173.1	134.1	19.6	247.0	573.7	132.3
	MH^4 (Single)	169.5	133.8	22.3	163.1	488.7	112.7
	HDR-mcd	140.6	148.9	19.2	124.1	432.8	99.8
	MH^4	158.8	151.5	18.3	160.4	489.0	112.7
	HDR-mcv (MSV)	87.1	139.6	10.7	81.3	318.6	73.5
	SAAV	90.1	44.2	8.3	64.0	206.6	47.6
	HDR-mcv	81.6	123.3	8.2	64.2	277.3	63.9
COPY (250)	SR	7441.3	2074.8	2054.7	11465.6	23036.4	100.0
	RDR	9936.0	761.9	1192.2	17548.0	29438.1	127.8
	MH^4 (Single)	9885.3	2165.5	681.3	13581.8	26313.8	114.2
	HDR-mcd	3278.2	2275.7	907.4	4150.4	10611.7	46.1
	MH^4	3837.0	637.1	354.9	6105.0	10933.9	47.5
	HDR-mcv (MSV)	3822.6	617.2	356.6	5994.6	10791.0	46.8
	HDR-mcv	1238.4	763.6	381.0	1922.9	4305.9	18.7
PARKER (10)	SR	20.6	1.2	29.5	19.4	70.7	100.0
	RDR	30.2	1.5	4.8	37.2	73.6	104.1
	MH^4 (Single)	40.9	2.1	15.4	37.0	95.4	135.0
	HDR-mcd	26.6	1.2	11.3	22.8	62.0	87.7
	MH^4	28.1	1.6	8.0	28.3	65.9	93.2
	HDR-mcv (MSV)	11.5	0.7	5.0	12.1	29.2	41.3
	HDR-mcv	9.3	0.6	5.9	8.3	24.2	34.2

Table 6.13: The results of clock and supply voltage assignment and the number of clock trees.

DCT

Method	$CF = 1 (= 1.5\,\text{ns})$			$CF = 2$			$CF = 4$			$\#CT$
	v_h	v_m	v_l	v_h	v_m	v_l	v_h	v_m	v_l	
SR	1									1
RDR	9									1
MH^4 (Single)	8									1
HDR-mcd	2			6						2
MH^4	2	6								2
HDR-mcv (MSV)			8							1
HDR-mcv			1			5			1	3

EWF3

Method	$CF = 1 (= 1.5\,\text{ns})$			$CF = 2$			$CF = 8$			$CF = 64$			$\#CT$
	v_h	v_m	v_l	v_h	v_m	v_l	v_h	v_m	v_l	v_h	v_m	v_l	
SR	1												1
RDR	8												1
MH^4 (Single)	8												1
HDR-mcd	3						3						2
MH^4	2	3	3										3
HDR-mcv (MSV)			5										1
HDR-mcv			1			3						3	3

FIR

Method	$CF = 1 (= 1.5\,\text{ns})$			$CF = 2$			$CF = 4$			$\#CT$
	v_h	v_m	v_l	v_h	v_m	v_l	v_h	v_m	v_l	
SR	1									1
RDR	9									1
MH^4 (Single)	8									1
HDR-mcd	1			7						2
MH^4		3	6							2
HDR-mcv (MSV)			8							1
HDR-mcv				1			4			2

JACOBI

Method	$CF = 1 (= 1.5\,\text{ns})$			$CF = 2$			$\#CT$
	v_h	v_m	v_l	v_h	v_m	v_l	
SR	1						1
RDR	8						1
MH^4 (Single)	6						1
HDR-mcd	3			4			2
MH^4	5	1	1				3
HDR-mcv (MSV)			6				1
SAAV			5				1
HDR-mcv			5				1

COPY

Method	$CF = 1 (= 1.5\,\text{ns})$			$CF = 2$			$CF = 4$			$CF = 128$			$\#CT$
	v_h	v_m	v_l	v_h	v_m	v_l	v_h	v_m	v_l	v_h	v_m	v_l	
SR	1												1
RDR	16												1
MH^4 (Single)	14												1
HDR-mcd	1			2			9			2			4
MH^4	1	1	12										3
HDR-mcv (MSV)			14										1
HDR-mcv			1			2			9			2	4

PARKER

Method	$CF = 1 (= 1.5\,\text{ns})$			$CF = 2$			$\#CT$
	v_h	v_m	v_l	v_h	v_m	v_l	
SR	1						1
RDR	2						1
MH^4 (Single)	5						1
HDR-mcd	4			2			2
MH^4	1	3	1				3
HDR-mcv (MSV)			3				1
HDR-mcv			3				1

Table 6.14: Average energy consumption normalized based on SR.

Tech.	Method	Energy ratio[%]
–	SR	100.0
–	RDR	117.4
–	MH^4 (Single)	102.2
MCD	HDR-mcd	73.9
MSV	HDR-mcv (MSV)	47.1
MSV+MCD	HDR-mcv	34.2

Chapter 7

Conclusion

In this dissertation, I propose new floorplan-driven SoC architectures to which energy-efficient LSI design techniques, such as multiple supply voltages technique (MSV), dynamic multiple supply voltages technique (DMSV), and multiple clock domains technique (MCD), are easily applicable. Furthermore, HLS algorithms are proposed for energy reduction based on the proposed architectures. The proposed algorithms can reflect floorplanning information in HLS by using iterative synthesis flow. By using a floorplanning result, interconnection delay and energy consumption can be estimated, and then optimized supply voltages and/or clock periods can be assigned for energy reduction.

The proposed HDR architecture divides chip area into several partitions called huddles. Huddles enable us to estimate interconnection delay effects easily and apply supply voltages effectively. The proposed iteration based HLS algorithm can consider interconnection dela   lly apply MSV. The basic algorithm is shown in Chapter 3. Experimental results show that the proposed method achieves 22.4% energy-saving compared with the conventional methods.

The basic HLS algorithm for HDR architecture in Chapter 3, however, has two problems, so the proposed MH^4 for HDR architecture is newly developed and enables us to quickly obtain solutions. The proposed virtual area estimation, virtual area adaptation, and foorplanning-directed huddling resolve the problems of the algorithm in Chapter 3. Experimental results show that the proposed algorithm achieves 29.1% run-time-saving compared with the algorithm proposed in Chapter 3, and successfully obtains a solution which cannot be obtained by the algorithm proposed in Chapter 3.

The proposed AVHDR architecture is introduced for applying DMSV. The proposed SAAV can assign low supply voltages to non-critical operations and can cut off leakage power through PG. Experimental results show that the proposed method achieves 43.9% energy-saving compared with the conventional methods.

The proposed HDR-mcd architecture can apply periodically all-in-phase based MCD. The SAMCID can consider the inter-domain synchronization which require interconnection delay and also realize efficient clock frequency selection and clock domain division by assigning an appropriate clock period. Experimental results show that the proposed method which only considers MCD achieves 32.5% energy-saving compared with the conventional methods. Furthermore, the proposed method which can apply MCD and MSV simultaneously achieves 57.0% energy-saving compared with the conventional methods.

In the future work, DMSV and MCD should be applied simultaneously. When huge leakage power of a functional unit is cut off through PG, we should not assign long clock period to the functional unit and associated registers with MCD. On the other hand, when low clock frequency is assigned to a huddle, the switch transistor control faster than clock frequency is impossible. Therefore, we should compare the effects of PG and MCD and judge which is better between the two methods for energy reduction.

Clock period selection and allocation problem are also the future work. For example, multiplexers can be decreased by adding extra functional units and/or registers. Since multiplexers consume huge area and energy consumption in recent LSI design, a high-performance allocation algorithm may realize further energy reduction. Process variation and soft-error problems are becoming major problems as device feature size decreases. Realizing an HLS algorithm for robust SoC is also the future work.

Acknowledgment

First of all, I deeply appreciate Professor Nozomu Togawa at the department of Computer Science and Engineering of Waseda University who gives me the considerable guidance. He teaches me not only the research techniques but also the manners of researcher. The attitude of researcher which is learned from him will be helpful for my research life.

I also appreciate Professor Masao Yanagisawa at the department of Electronic and Photonic Systems of Waseda University and Professor Shinji Kimura at the Graduate School of Information, Production, and Systems of Waseda University. They always give me a lot of suggestions and opinions which strongly advance my research.

I also appreciate Professor Kimiyoshi Usami at the department of Information Science and Engineering of Shibaura Institute of Technology. The technical advice from him exceedingly improves my research and this dissertation.

I am deeply grateful to Professor Youhua Shi for his advice and support to my research life.

I also thank the students in Togawa laboratory and Yangisawa laboratory for their kindness.

This dissertation was supported partially by JSPS KAKENHI Grand-in-Aid for JSPS Fellows.

Finally, I deeply appreciate my family for their many years of support.

References

[1] H. Akasaka, S. Abe, M. Yanagisawa, and N. Togawa, “Energy-efficient high-level synthesis for hdr architectures with clock gating based on concurrency-oriented scheduling,” *IPSJ Trans. on System LSI Design Methodology*, vol. 6, pp. 101–111, 2013.

[2] J. An, H. Park, and Y. Kim, “Level up/down converter with single power-supply voltage for multi-vdd systems,” *J. of Semiconductor Technology and Science*, vol. 10, no. 1, pp. 55–60, 2010.

[3] T. D. Burd, T. A. Pering, A. J. Stratakos, and R. W. Brodersen, “A dynamic voltage scaled microprocessor system,” *IEEE J. of Solid-State Circuits*, vol. 35, no. 11, pp. 1571–1580, 2000.

[4] J. Chang and M. Pedram, “Energy minimization using multiple supply voltages,” *IEEE Trans. on VLSI Systems*, vol. 5, no. 4, pp. 436–443, 1997.

[5] D. M. Chapiro, *Globally-asynchronous locally-synchronous systems.* PhD thesis, Stanford University, 1984.

[6] D. Chen, J. Cong, and J. Xu, “Optimal simultaneous module and multivoltage assignment for low power,” *ACM Trans. on Design Automation of Electronic Systems*, vol. 11, no. 2, pp. 362–386, 2006.

[7] E. Choi, C. Shin, T. Kim, and Y. Shin, “Power-gating-aware high-level synthesis,” in *Proc. of ISLPED ’08*, pp. 39–44, 2008.

[8] J. Cong, Y. Fan, G. Han, X. Yang, and Z. Zhang, “Architecture and synthesis for on-chip multicycle communication,” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 4, pp. 550–564, 2004.

[9] J. Cong, Y. Fan, and J. Xu, “Simultaneous resource binding and interconnection optimization based on a distributed register-file microarchitecture,” *ACM Trans. on Design Automation of Electronic Systems*, vol. 14, no. 3, pp. 1–31, 2009.

[10] D. Dal and N. Mansouri, "Power optimization with power islands synthesis," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 7, pp. 1025–1037, 2009.

[11] R. Gonzalez and M. Horowitz, "Energy dissipation in general purpose microprocessors," *IEEE J. of Solid-State Circuits*, vol. 31, no. 9, pp. 1277–1284, 1996.

[12] International technology roadmap for semiconductors (ITRS), 2003. http://www.itrs.net/.

[13] International technology roadmap for semiconductors (ITRS), 2005. http://www.itrs.net/.

[14] International technology roadmap for semiconductors (ITRS), 2007. http://www.itrs.net/.

[15] International technology roadmap for semiconductors (ITRS), 2009. http://www.itrs.net/.

[16] International technology roadmap for semiconductors (ITRS), 2011. http://www.itrs.net/.

[17] J. Jeon, D. Kim, D. Shin, and K. Choi, "High-level synthesis under multi-cycle interconnect delay," in *Proc. of ASP-DAC '01*, pp. 662–667, 2001.

[18] M. Johnson and K. Roy, "Datapath scheduling with multiple supply voltages and level converters," *ACM Trans. on Design Automation of Electronic Systems*, vol. 2, no. 3, pp. 227–248, 1997.

[19] D. Kim, J. Jung, S. Lee, J. Jeon, and K. Choi, "Behavior-to-placed rtl synthesis with performance-driven placement," in *Proc. of ICCAD '01*, pp. 320–325, 2001.

[20] S. Kohara, Y. Shi, N. Togawa, M. Yanagisawa, and T. Ohtsuki, "Codama: An xml-based framework to manipulate control data flow graphs," in *Proc. of SASIMI '07*, pp. 545–549, 2007.

[21] A. Kumar and M. Bayoumi, "Multiple voltage-based scheduling methodology for low power in the high level synthesis," in *Proc. of ISCAS 1999*, vol. 1, pp. 371–374, 1999.

[22] W. Lee, H. Liu, and Y. Chang, "Voltage island aware floorplanning for power and timing optimization," in *Proc. of ICCAD '06*, pp. 389–394, 2006.

[23] G. Lhairech-Lebreton, P. Coussy, and E. Martin, "Hierarchical and multiple-clock domain high-level synthesis for low-power design on fpga," in *Proc. of FPL '10*, pp. 464–468, 2010.

[24] F. Li, Y. Lin, and L. He, "Fpga power reduction using configurable dual-vdd," in *Proc. of DAC '04*, pp. 735–740, 2004.

[25] Y. Lin, C. Hwang, and A. Wu, "Scheduling techniques for variable voltage low power designs," *ACM Trans. on Design Automation of Electronic Systems*, vol. 2, no. 2, pp. 81–97, 1997.

[26] A. Manzak and C. Chakrabarti, "A low power scheduling scheme with resources operating at multiple voltages," *IEEE Trans. on VLSI Systems*, vol. 10, no. 1, pp. 6–14, 2002.

[27] S. Mohanty and N. Ranganathan, "Simultaneous peak and average power minimization during datapath scheduling," *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 52, no. 6, pp. 1157–1165, 2005.

[28] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "VLSI module placement based on rectangle-packing by the sequence-pair," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 12, pp. 1518–1524, 1996.

[29] NCSU CBL. http://www.cbl.ncsu.edu/benchmarks/.

[30] K. Nose, A. Shibayama, H. Kodama, M. Mizuno, M. Edahiro, and N. Nishi, "Deterministic inter-core synchronization with periodically all-in-phase clocking for low-power multi-core socs," in *ISSCC Dig. Tech. Papers*, pp. 296–599, 2005.

[31] A. Ohchi, S. Kohara, N. Togawa, M. Yanagisawa, and T. Ohtsuki, "Floorplan-driven high-level synthesis for distributed/shared-register architectures," *IPSJ Trans. on System LSI Design Methodology*, vol. 1, pp. 78–90, 2008.

[32] A. Ohchi, N. Togawa, M. Yanagisawa, and T. Ohtsuki, "Floorplan-aware high-level synthesis for generalized distributed-register architectures," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 92, no. 12, pp. 3169–3179, 2009.

[33] M. Putic, L. Di, B. Calhoun, and J. Lach, "Panoptic dvs: A fine-grained dynamic voltage scaling framework for energy scalable cmos design," in *Proc. of ICCD '09*, pp. 491–497, 2009.

[34] S. Raje and M. Sarrafzadeh, "Variable voltage scheduling," in *Proc. of ISLPED '95*, pp. 9–14, 1995.

[35] M. Sarrafzadeh and S. Raje, "Scheduling with multiple voltages under resource constraints," in *Proc. of ISCAS 1999*, vol. 1, pp. 350–353, 1999.

[36] N. Seki, L. Zhao, J. Kei, D. Ikebuchi, Y. Kojima, Y. Hasegawa, H. Amano, T. Kashima, S. Takeda, T. Shirai, M. Nakata, K. Usami, T. Sunata, J. Kanai, M. Namiki, M. Kondo, and H. Nakamura, "A fine-grain dynamic sleep control scheme in mips r3000," in *Proc. of ICCD '08*, pp. 612–617, 2008.

[37] I. Shin, S. Paik, and Y. Shin, "Register allocation for high-level synthesis using dual supply voltages," in *Proc. of DAC '09*, pp. 937–942, 2009.

[38] W. Shiue and C. Chakrabarti, "Low-power scheduling with resources operating at multiple voltages," *IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 47, no. 6, pp. 536–543, 2000.

[39] S. Tanaka, M. Yanagisawa, T. Ohtsuki, and N. Togawa, "A fault-secure high-level synthesis algorithm for rdr architectures," *IPSJ Trans. on System LSI Design Methodology*, vol. 4, pp. 150–165, 2011.

[40] C. Tran, H. Kawaguchi, and T. Sakurai, "Low-power high-speed level shifter design for block-level dynamic voltage scaling environment," in *Proc. of ICICDT 2005*, pp. 229–232, 2005.

[41] K. Usami, M. Igarashi, F. Minami, T. Ishikawa, M. Kanzawa, M. Ichida, and K. Nogami, "Automated low-power technique exploiting multiple supply voltages applied to a media processor," *IEEE J. of Solid-State Circuits*, vol. 33, no. 3, pp. 463–472, 1998.

[42] K. Usami, T. Shirai, T. Hashida, H. Masuda, S. Takeda, M. Nakata, N. Seki, H. Amano, M. Namiki, M. Imai, M. Kondo, and H. Nakamura, "Design and implementation of fine-grain power gating with ground bounce suppression," in *Proc. of VLSID 2009*, pp. 381–386, 2009.

[43] A. Vittal and M. Marek-Sadowska, "Low-power buffered clock tree design," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 9, pp. 965–975, 1997.

[44] K. Wakabayashi and T. Yoshimura, "A resource sharing and control synthesis method for conditional branches," in *Proc. of ICCAD '89*, pp. 62–65, 1989.

[45] N. Weste and D. Harris, *CMOS VLSI Design.* Addison-Wesley Publishing Company, fourth ed., 2010.

[46] X. Xing and C. Jong, “Floorplan-driven multivoltage high-level synthesis,” *VLSI Design*, vol. 2009, pp. 1–10, 2009.

[47] H. Yang and L. Dung, “On multiple-voltage high-level synthesis using algorithmic transformations,” in *Proc. of ASP-DAC ’05*, pp. 872–876, 2005.

[48] H. Yang and L. Dung, “Algorithmic transformations and peak power constraint applied to multiple-voltage low-power VLSI signal processing,” *WSEAS Trans. on Signal Processing*, vol. 3, no. 12, pp. 479–486, 2007.

List of Publications

論文（学術誌原著論文）

1. ◯ **S. Abe**, K. Usami, M. Ynagisawa, and N. Togawa, “Floorplan driven architecture and high-level synthesis algorithm for dynamic multiple supply voltages,” *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 96, no. 12, pp. 2597–2611, Dec., 2013.

2. H. Akasaka, **S. Abe**, M. Yanagisawa, and N. Togawa, “Energy-efficient high-level synthesis for hdr architectures with clock gating based on concurrency-oriented scheduling,” *IPSJ Trans. on System LSI Design Methodology*, vol. 6, pp. 101–111, Aug., 2013.

3. ◯ **S. Abe**, Y. Shi, M. Yanagisawa, and N. Togawa, “Mh^4 : multiple-supply-voltages aware high-level synthesis for high-integrated and high-frequency circuits for hdr architectures,” *IEICE Electronics Express*, vol. 9, no. 17, pp. 1414–1422, Sept., 2012.

4. ◯ **S. Abe**, M. Yanagisawa, and N. Togawa, “Energy-efficient high-level synthesis for hdr architectures,” *IPSJ Transactions on System LSI Design Methodology*, vol. 5, pp. 106–117, Aug., 2012.

国際会議

(招待講演・口頭発表)

1. ◯ **S. Abe** and N. Togawa, “Floorplan-driven architecture and high-level synthesis algorithm for energy optimization,” in *Proceedings of the 29th International Technical Conference on Circuit/Systems Computers and Communications (ITC-CSCC 2014)*, pp. 733–736, Jul. 2014.

(口頭発表)

1. ◯ **S. Abe**, Y. Shi, K. Usami, M. Yanagisawa, and N. Togawa, “An energy-efficient high-level synthesis algorithm incorporating interconnection delays

and dynamic multiple supply voltages," in *Proceedings of the 2013 International Symposium on VLSI Design, Automation and Test (VLSI-DAT 2013)*, pp. 54–57, Apr. 2013.

2. ◯ **S. Abe**, M. Yanagisawa, and N. Togawa, "An Energy-efficient High-level Synthesis Algorithm for Huddle-based Distributed-Register Architectures," in *Proceedings of 2012 IEEE International Symposium on Circuits and Systems (ISCAS 2012)*, pp. 576–579, May 2012.

(ポスター発表)

1. ◯ **S. Abe**, Y. Shi, K. Usami, M. Yanagisawa, and N. Togawa, "An energy-efficient high-level synthesis algorithm incorporating interconnection delays and dynamic multiple supply voltages," in *Proceedings of the 2013 International Symposium on VLSI Design, Automation and Test (VLSI-DAT 2013)*, pp. 54–57, Apr. 2013.

国内学会（査読付き）

1. ◯ 阿部晋矢, 史又華, 宇佐美公良, 柳澤政生, 戸川望, "フロアプラン統合化アーキテクチャを対象とした複数クロックドメインおよび複数電源電圧による低電力化高位合成手法," 情報処理学会 DA シンポジウム 2013, pp. 139–144, Aug. 2013.
2. 赤坂宏行, 阿部晋矢, 柳澤政生, 戸川望, "フロアプラン統合化アーキテクチャを対象とした複数クロックドメインおよび複数電源電圧による低電力化高位合成手法," 情報処理学会 DA シンポジウム 2013, pp. 43–48, Aug. 2013.
3. ◯ 阿部晋矢, 史又華, 宇佐美公良, 柳澤政生, 戸川望, "Hdr-mcd を対象としたマルチクロックドメイン指向の低電力化高位合成手法," 第 26 回 回路とシステムワークショップ, pp. 185–190, Jul. 2013.
4. ◯ 阿部晋矢, 宇佐美公良, 柳澤政生, 戸川望, "動的複数電源電圧およびフロアプラン統合化アーキテクチャを対象とした低電力化高位合成手法," 情報処理学会 DA シンポジウム 2012, pp. 163–168, Aug. 2012.
5. ◯ 阿部晋矢, 柳澤政生, 戸川望, "高集積かつ高周波な回路に対応した複数電源電圧指向の高位合成手法," 第 25 回 回路とシステムワークショップ, pp. 160–165, Jul. 2012.
6. ◯ 阿部晋矢, 柳澤政生, 戸川望, "複数電源電圧および複数サイクルレジスタ間通信指向の低電力化高位合成手法," 情報処理学会 DA シンポジウム 2011, pp. 21–26, Aug. 2011.

国内学会 (査読無し)

1. ◯ 阿部晋矢, 史又華, 宇佐美公良, 柳澤政生, 戸川望, “Hdr-mcd を対象としたクロックエネルギー優位な高位合成と実験評価,” 信学技報, vol. 113, pp. 263–268, Nov. 2012.

2. ◯ 阿部晋矢, 史又華, 柳澤政生, 戸川望, “フロアプランを考慮したマルチクロックドメイン指向の低電力化高位合成手法,” 情報処理学会研究報告 2013-SLDM-160(20), pp. 1–6, Mar. 2013.

3. ◯ 阿部晋矢, 史又華, 宇佐美公良, 柳澤政生, 戸川望, “Saav:avhdr アーキテクチャを対象とした動的複数電源電圧指向の低電力化高位合成手法,” 信学技報 , vol. 112, pp. 135–140, Nov. 2012.

4. ◯ 阿部晋矢, 柳澤政生, 戸川望, “Hdr アーキテクチャを対象とした高速かつ効率的な複数電源電圧指向の高位合成手法,” 信学技報, vol. 112, pp. 7–12, May 2012.

5. ◯ 阿部晋矢, 柳澤政生, 戸川望, “Hdr アーキテクチャを対象とした複数電源電圧指向の低電力化高位合成手法,” 情報処理学会研究報告 2011-SLDM-152(17), pp. 1–6, Oct. 2011.

業績賞等

1. 2014 年 11 月 情報処理学会デザインガイアポスター賞.
2. 2014 年 8 月 情報処理学会優秀発表学生賞.
3. 2014 年 3 月 情報処理学会山下記念研究賞.
4. 2013 年 8 月 情報処理学会優秀発表学生賞.
5. 2012 年 8 月 情報処理学会優秀発表学生賞.
6. 2012 年 5 月 電子情報通信学会集積回路研究専門委員会優秀若手講演賞.

日本学術振興会 科学研究費補助金

1. 日本学術振興会特別研究員奨励費, “低消費電力 LSI 実現へ向けた高位・物理統合設計技術の開発,” 2013-2014 年度, 総額 220 万円 (2013 年度:110 万円, 2014 年度:110 万円).