Floorplan-Aware High-Level Synthesis Algorithms and their Acceleration by Ising Computations

February 2018

Kotaro TERADA

寺田 晃太朗

Floorplan-Aware High-Level Synthesis Algorithms and their Acceleration by Ising Computations

フロアプラン指向高位合成手法と イジング計算機応用に関する研究

February 2018

Waseda University

Graduate School of Fundamental Science and Engineering

Department of Computer Science and Communications Engineering,

Research on Information System Design

Kotaro TERADA

寺田 晃太朗

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

System LSI or system-on-a-chip (SoC) is an essential device in today's highly sophisticated information society. System LSI is already used widely in the world such as personal computers, servers, mobile phones, automobiles, and consumer electronics. In the Internet of thing (IoT) era, every "things" about to become informatization and systematized. Many SoCs with higher information processing are expected to be more and more needed. In these kinds of system designs, it is required to design a high-performance and small area of a system LSI with a lower cost and a shorter time period.

Based on the background above, high-level synthesis (HLS) [21] has played a significant and essential role as a promising electronic design automation (EDA) technique nowadays. HLS synthesizes abstract application descriptions such as C, C++, Java, and Python into register transfer level (RTL) descriptions. Since we can begin a system design with a higher level of abstraction, it is expected to reduce a cost of design such as design time (Time-to-Market) and design errors.

The main input to HLS is a behavioral description, and HLS outputs FSMD (Finite State Machine with Datapath) which is composed of a datapath and a controller (FSM). An input behavioral description is represented as a control data flow graph (CDFG) where nodes correspond to operations and edges correspond to data-flows. An HLS algorithm typically solves (1) Scheduling problem: assigns every operation in the CDFG to a control step, (2) Allocation problem: determines the number and the type of functional units (FUs) to be used, (3) Binding problem: assigns operations and variables to FUs and registers, respectively.

Many HLS methods and tools have been proposed so far including ones for academic purposes and ones for commercial purposes. For example, Xilinx, Inc. provides Vivado Design Suite [64] for free (targeting for the limited types of FPGA). The group of

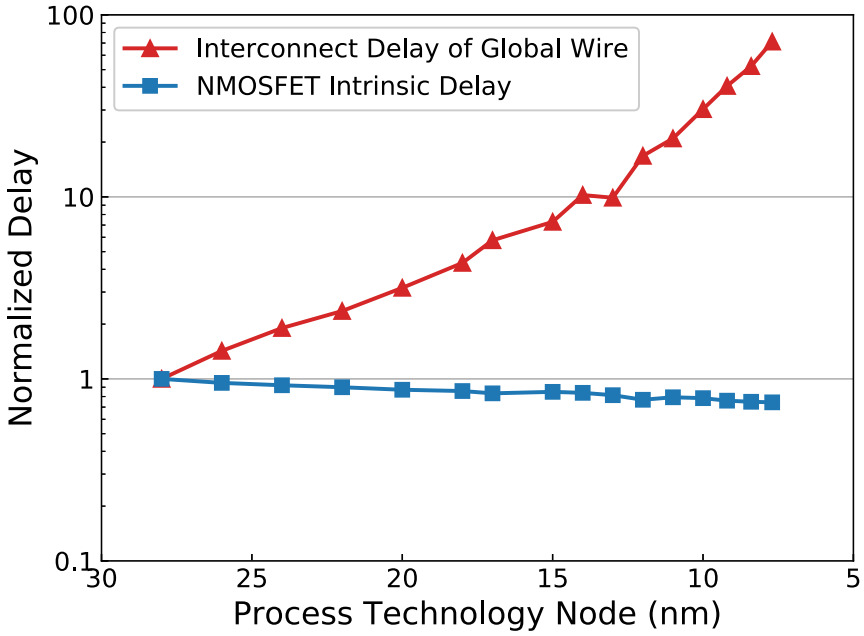


Figure 1.1: Interconnection delays vs. gate delays based on ITRS 2013 [30].

University of Toronto has developed LegUp [9, 10, 60] as an open-source software. Synthesijer [41] is also an open-source HLS tool which synthesizes Java codes and generates VHDL and Verilog HDL. Polyphony [53] is also an open-source HLS tool which synthesizes Python codes.

Field-programmable gate array (FPGA) comes into fashion nowadays. For example, ZYNQ [65] by Xilinx, Inc. is a cheap SoC and it includes ARM processors as well as FPGA. PYNQ [66] is a board which contains a ZYNQ SoC[1]. Not only commercial designers and experts but any users can use FPGA easily and cheaply. HLS becomes more and more essential nowadays.

However, in the deep-submicron era, interconnection delays are not negligible even in high-level synthesis. Figure 1.1 shows the relative comparison between interconnection delay and gate delay. The figure is based on the predicted data for 2013 to 2028 by International Technology Roadmap for Semiconductors (ITRS) 2013 [30]. The relations of the process technology node and (i) RC delays of global wires (interconnection delays) and (ii) delays of NMOSFET (gate delays) are plotted. All values are normalized by the values of 2013. The horizontal axis shows process technology node and the vertical axis shows relative delays. In Fig. 1.1, we can see that the interconnection delay is

[1]The author and his colleagues have developed ZYNQ-based puzzle solver systems using Vivado HLS for the "Algorithm Design Contest" as in ⟨7⟩ and ⟨9⟩, and won the prizes as in ⟨15⟩, ⟨16⟩, and ⟨17⟩

relatively increasing against the gate delay. Since the process technology node is getting smaller nowadays, the gap between the interconnection delays and the gate delays is increasing more and more. In the deep-submicron era, the interconnection delays are the dominant among all delays. To design a high performance VLSI circuit (i.e., circuit with low latency) is a basic requirement. However, the issue caused by interconnection delays prevents it seriously. In this decade, to cope with this problem, distributed-register and -controller architectures (DR architectures) along with their HLS algorithms have been proposed as in [2, 3, 15, 18, 28, 36, 45] against conventional shared-register architectures (SR architectures). RDR architecture [15] is one of DR architectures. DR architecture is divided into clusters. In DR architecture, registers, FUs, and controllers are placed distributed onto the clusters, and we can leave the interconnection delays between registers and FUs being much smaller than those of SR architectures.

To design a high performance VLSI circuit, "operation chaining" and "bitwidth-based optimization" are reliable techniques[2]. HLS algorithms with operation chainings and/or bitwidth optimization are proposed so far as in [13,18,20,35,54,56,75]. However, since these works do not take the interconnection delays into account, we cannot use them for DR architectures. [17] proposes a bitwidth-aware HLS algorithm considering interconnection delays, however, it focuses on the area minimization and does not optimize the latency.

Taking account the interconnection delays in HLS makes the optimizations difficult since we have to deal with the delays between modules (such as FUs and registers). In the first half of this dissertation (Chapter 2, Chapter 3, and Chapter 4), we propose performance-driven floorplan-aware HLS algorithms targeting RDR architectures. The objective is to minimize the latency (or maximize the performance). Table 1.1 shows the survey on HLS algorithms with operation chainings and/or bitwidth optimization targeting SR and DR architectures and relationship between this dissertation and other works. Obviously from the table, to the best of our knowledge, Chapter 2 and Chapter 3 are the first HLS algorithms with operation chainings targeting RDR architectures, and Chapter 4 is the first bitwidth-aware HLS algorithm using operation chainings targeting RDR architectures.

Even if we optimize the system design through HLS, the floorplanning depends on the metaheuristic algorithm, typically simulated annealing (SA) [32]. Thus, the floorplanning remains the bottleneck in both speed and scalability.

Floorplanning of modules has been a significant role in VLSI design automation and it can be formulated as the "Rectangles Packing Problem." Novel physical Ising model-based computers (annealing machines), which are the type of a non-von Neumann computer, have been recently expected to solve combinatorial optimization problems

[2]The details of these techniques are described in Chapter 2, Chapter 3, and Chapter 4.

Table 1.1: Survey on HLS algorithms with operation chainings and/or bitwidth optimization targeting SR and DR architectures.

	SR architectures	**DR architectures**
uniform bitwidth	many methods	[15] and many others
bitwidth-aware	[35] and others	[17]
w/ operation chainings	many methods	**Chapter 2** and **Chapter 3**
bitwidth-aware w/ operation chainings	[20]	**Chapter 4**

efficiently. CMOS annealing machine [70] is one of them. In those studies, a combinatorial optimization problem is mapped onto a theoretical magnetic model in statistical mechanics called Ising model. Ising model-based computers search the ground-state of the Ising model, which corresponds to the optimal solution of the original combinatorial optimization problem.

Some physical Ising model-based computers have been developed such as D-Wave quantum annealing machine [7, 31], CMOS annealing machines [70, 73], and FPGA-based annealing machine by Fujitsu Laboratories Ltd. [59].

To implement an Ising model to physical annealing machines, we need to convert an Ising model with an arbitrary topology into an Ising model with the physical topology. This flow is called embedding. If we achieve the efficient embedding algorithm, we can increase the problem size implemented onto the annealing machines and may have better results. Embedding the Ising model to the physical models plays an important role in the programming flow of annealing machines.

The floorplanning algorithms in Chapter 2, Chapter 3 and Chapter 4 are the bottleneck in both speed and scalability as mentioned above. These annealing machines can be used to accelerate these bottlenecks. In the last half of this dissertation (Chapter 5 and Chapter 6), acceleration of the floorplanning problem by annealing machines is proposed. Solving relatively simple problems by annealing machines are studied so far. However, not so many works focus on more practical problems such as the floorplanning problem. This dissertation also contributes a large impact for the practical uses of annealing machines.

1.2 Dissertation Overview

In this dissertation, we propose three floorplan-aware performance-driven HLS algorithms targeting RDR architectures to cope with the increasing interconnection delays. We also propose an embedding method of fully-connected Ising model onto 20k-spin

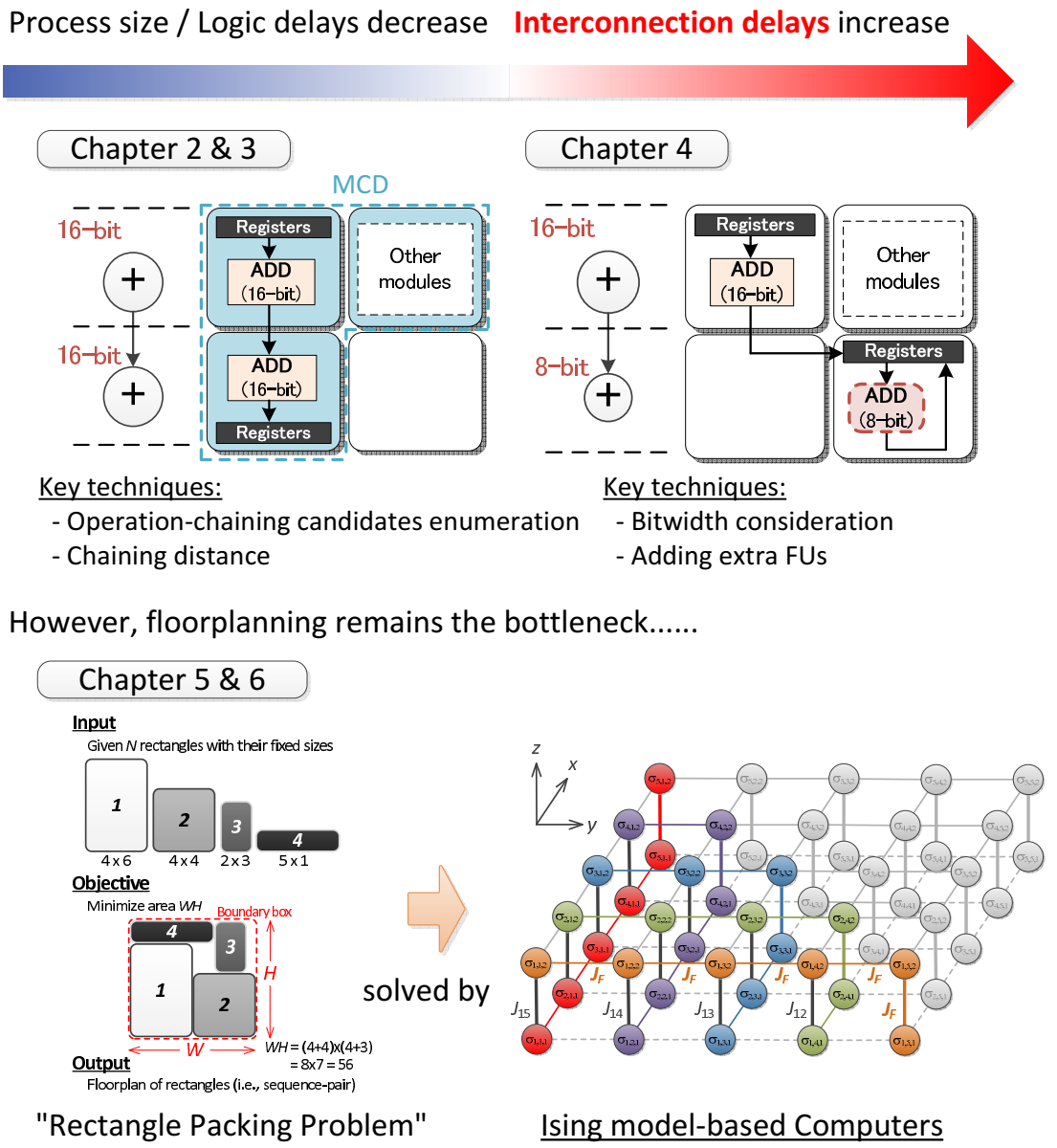


Figure 1.2: Overview of this dissertation.

CMOS annealing machines followed by proposing a mapping of the floorplan problem for Ising model-based computers to deal with the bottleneck in the floorplanning. Figure 1.2 shows the overview of this dissertation.

This dissertation is organized as follows:

Chapter 2 [A Floorplan-Aware High-Level Synthesis Algorithm with Operation Chainings Based on Inter-Island Distance] proposes a high-level synthesis algorithm using operation chainings which reduces the overall latency targeting RDR architectures.

The proposed algorithm consists of three steps: The first step enumerates candidate operations for chaining. The second step introduces *maximal chaining distance (MCD)*, which gives the maximal allowable inter-island distance on RDR architecture between chaining candidate operations. The last step performs list-scheduling and binding simultaneously based on the results of the two preceding steps. The proposed algorithm enumerates feasible chaining *candidates* and selects the best ones for RDR architecture. Experimental results show that our proposed algorithm reduces the latency by up to 40.0% compared to the original approach, and by up to 25.0% compared to a conventional approach. The proposed algorithm also reduces the number of registers and the number of multiplexers compared to the conventional approaches in some cases.

Chapter 3 [A Floorplan-Aware High-Level Synthesis Algorithm with Multiple-Operation Chainings Based on Path Enumeration] proposes a floorplan-driven high-level synthesis algorithm using multiple-operation chainings composed of two or more operations, and reduce the overall latency targeting RDR architecture. The proposed algorithm enumerates multiple-operation-chaining path candidates before performing scheduling/binding. Based on them, we find out optimal ones taking into account RDR floorplan information. Experimental results show that our algorithm reduces the latency by up to 30.4% compared to a conventional algorithm, and reduces the latency by up to 24.4% compared to the algorithm proposed in Chapter 2, but is only effective to the limited benchmarks.

Chapter 4 [A Floorplan-Driven Bitwidth-Aware High-Level Synthesis Algorithm Using Operation Chainings] proposes a bitwidth-aware high-level synthesis algorithm using operation chainings targeting RDR architectures. Our proposed algorithm optimizes bitwidths of functional units and utilizes the vacant islands by adding some extra functional units to realize effective operation chainings to generate high-performance circuits without increasing the total area. Experimental results show that our proposed algorithm reduces the latency by up to 47% compared to the algorithm proposed in Chapter 2 without area overheads by eliminating unnecessary bitwidths and adding efficient extra FUs for RDR architectures.

These algorithms above successfully reduce the latency compared to the conventional algorithms coping with the increasing interconnection delays. However, the SA-based floorplanning remains the bottleneck in both speed and scalability. To deal with the bottleneck above, we try to apply and accelerate the floorplanning problem to the forthcoming Ising model-based computers (annealing machines).

Chapter 5 [A Fully-Connected Ising Model Embedding Method for 20k-Spin CMOS Annealing Machines] proposes a fully-connected Ising model embedding method onto 20k-spin CMOS annealing machine, and prove that the ground state of the Ising models obtained from our proposed method is equivalent to that of the original Ising model. Experimental results effectively show that our proposed method embeds

Ising models using less physical spins compared to the de facto standard conventional method in the practical problem size, and that the probability of feasible solutions and the solution quality using our proposed method is better than those of the conventional method when solving practical combinatorial optimization problems.

Chapter 6 [Rectangle Packing Problem by Ising Computers] proposes a mapping of "Rectangle Packing Problem" for solving it by the Ising model-based computers. In our proposed mapping, a sequence-pair is represented on an Ising model and the energy function to obtain the optimal solution of the problem is constructed. Our proposed approach maps a "Rectangle Packing Problem" with N rectangles onto a $3N^3$-spin logical Ising model. Experimental results demonstrate that through the proposed approach we can solve the problem with nine rectangles at the maximum on a fully-connected annealing machine and the problem with three rectangles at the maximum on 20k-spin CMOS annealing machine.

Chapter 7 [Conclusion] summarizes this dissertation and gives some future works.

Chapter 2

A Floorplan-Aware High-Level Synthesis Algorithm with Operation Chainings Based on Inter-Island Distance[1]

2.1 Introduction

As process technologies advance in deep-submicron era, it comes to be able to produce highly integrated circuits. High-level synthesis (HLS) is one of the reliable solutions to this problem, which synthesizes register-transfer level circuits from abstract behavioral descriptions. HLS mainly consists of scheduling which assigns operation nodes to control steps, allocation which selects functional units (FUs) from libraries, and binding which binds operation nodes and variables to FUs and registers.

Highly integrated circuits cause interconnection delays to be relatively larger than gate delays, which has become a main concern even in HLS. Interconnection delays may exceed the clock period and a communication over an entire chip can no longer be possible.

To cope with this problem, regular-distributed-register architectures (RDR architectures) have been introduced in [15]. RDR architecture is one of the distributed-register (DR) architectures in which registers are distributed over a chip, while registers are concentrated in conventional shared-register (SR) architectures. RDR architecture is divided into *islands* and enables multi-cycle communications on a chip easily. Each island has the same size and then we can easily estimate interconnection delays on RDR architecture for its regularity. An HLS algorithm for RDR architecture called MCAS

[1]Technical contents in this chapter have been presented in the publications ⟨2⟩, ⟨4⟩, and ⟨14⟩.

was also proposed in [15].

We also have to deal with operation delays in HLS since every operation has a different delay. For example, an adder typically has a smaller delay than a multiplier. *Operation chainings* can solve this problem where we pack several data-depending operations into packed control steps and reduce the overall latency.

In this chapter, we propose an HLS algorithm using operation chainings for RDR architecture to reduce the overall latency. Our algorithm consists of three steps: The first step enumerates chaining candidates. The second step introduces *maximal chaining distance (MCD)*, which gives the maximal allowable inter-island distance on RDR architecture between chaining candidate operations. The last step performs list-scheduling and binding simultaneously based on the results of the two preceding steps. Our algorithm enumerates feasible chaining candidates and selects the best ones for RDR architectures, while in many other algorithms the candidates are given as assumptions. Experimental results show that our algorithm reduces the latency, the number of registers, the number of MUXs, compared to the conventional approaches.

The contributions of this chapter are:

(1) We realize the first floorplan-driven HLS with operation chainings which takes into account inter-island delays in RDR architecture.

(2) Experimental results successfully demonstrate that our HLS algorithm efficiently reduces the latency by applying operation chainings to the critical paths explicitly including interconnection delays, compared to the conventional approaches.

This chapter is organized as follows: Section 2.2 reviews related works; Section 2.3 describes our problem formulation; Section 2.4 proposes our HLS algorithm with operation chainings for RDR architecture; Section 2.5 shows experimental results; Section 2.6 gives several concluding remarks.

2.2 Related Works

Many HLS algorithms with operation chainings have been studied. In [34,39,40,49,52], an HLS problem using operation chainings is formulated as an integer linear programming (ILP) problem. In [40, 58], timing-variation-aware HLS algorithms are proposed which minimize the timing variations using operation chainings. In [42,43,47,48,51], schedulings and bindings using bit-level operation chainings are proposed. In [71], a scheduling algorithm based on list scheduling using operation chainings is proposed for instruction cell based reconfigurable system. Since the approach in [71] has been proposed for special-purpose architecture called instruction cell based reconfigurable system, we cannot apply this approach to general HLS, although it takes into account

interconnection delays between cells. In [74], an operation delay model for FPGA and a scheduling algorithm using chainings are proposed to reduce control steps. In [63], a heuristic scheduling algorithm using operation chainings is proposed. In many algorithms, chaining candidates are given as assumptions but they are not given in some algorithms. In [19, 67], template matchings are used to enumerate chaining candidates. In [74], chaining candidates are enumerated by calculating operation delay time sequentially and the operations on critical paths have high priorities to be chained. In [63], all feasible chaining patterns are first generated and then we reduce them which cannot reduce the latency using a cost function. Chaining candidates are searched greedy [38] or formulated as ILP [34, 52]. Most of these studies are, however, based on SR architecture and no HLS algorithms with operation chainings have been proposed for RDR architectures.

2.3 Problem Formulation

In this section, we define control-data flow graphs and RDR architecture. Then we define operation chainings on RDR architectures. After that we define our HLS problem.

2.3.1 Control-Data Flow Graphs

An input of HLS is a behavioral description represented by a control-data flow graph (CDFG). CDFG consists of data-flow graphs (DFGs) which represent operations and their data-flows and control-flow graphs (CFGs) which represent control-flows including conditional branches. For simplicity, we use a DFG in this chapter to explain our algorithm but we can extend it to CDFG easily as in our experimental results.

A DFG is usually represented by a directed graph. Let $G = (V, E)$ be a DFG where V is a set of operation nodes and E is a set of edges between them. An edge from a node v_1 to a node v_2 is denoted by e_{v_1,v_2}.

Let d_{fu} be a delay of an FU fu and d_{reg} be the sum of reading time and writing time of a register. We assume that an FU delay includes MUX delays.[2] MUX delays are given as a constant value. Let $dsum(fu)$ be the sum of the delays of reading data from a register, operation on an FU fu, and writing data to a register and then it is calculated by

$$dsum(fu) = d_{reg} + d_{fu}. \tag{2.1}$$

[2]We assume that both FU inputs and outputs have MUXs and total delay of them is given to each FU as a constant value. As in Section 2.5, its value is just 0.12ns and we expect that it does not affect too much. Note that register delay does not include MUX delays since they are included at FU output. In case that register-to-register data transfer is required as in Fig. 2.2(b), an interconnection delay between islands must be dominant and then we can ignore MUX delays in this case.

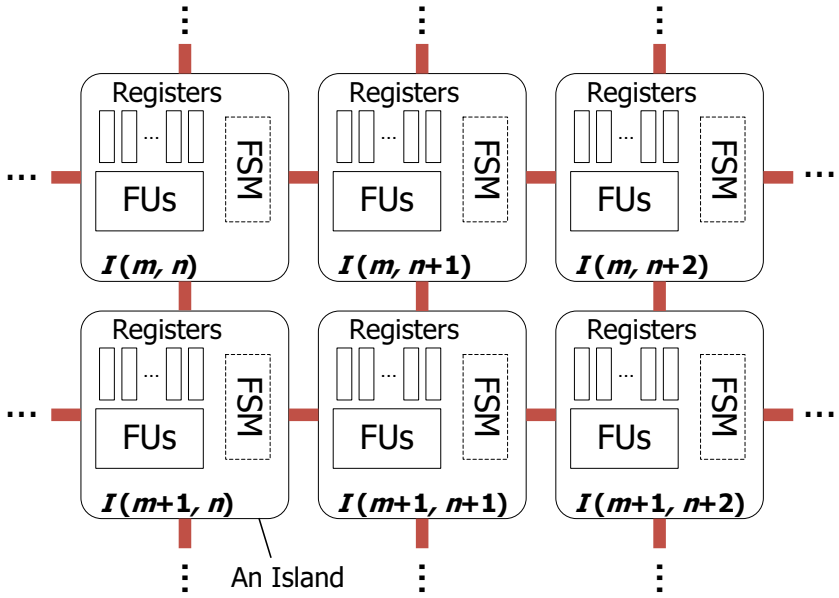


Figure 2.1: RDR architecture model.

In this chapter, for simplicity, each operation can be bound to one type of FUs, and an FU can execute only one type of operations. Therefore, each operation v has a unique delay $d(v)$. Let $EFU(v)$ be a set of FUs which can execute an operation v, which is given as an input.

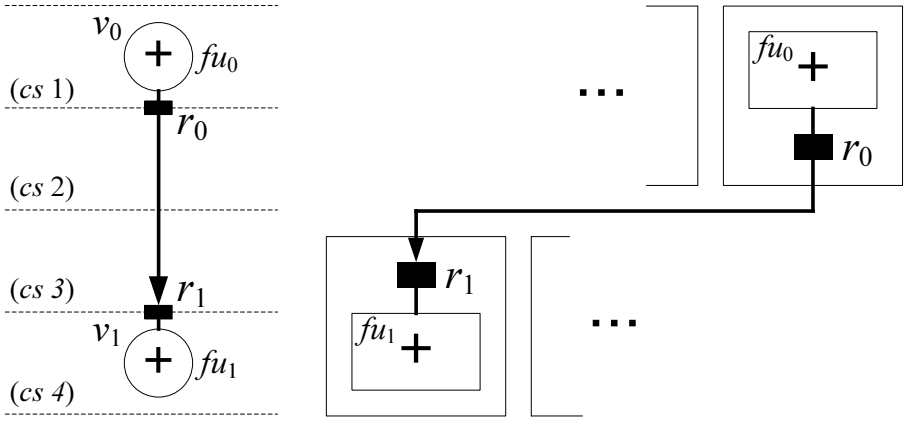
A set of the immediate predecessors and successors of $v \in V$ are denoted by $P(v)$ and $S(v)$, respectively. The nodes which have no predecessors are called primary inputs (PIs) and a set of them is denoted by PIs. The nodes which have no successors are called primary outputs (POs) and a set of them is denoted by POs. The level $\ell(v)$ of the operation node v is defined by:

$$\ell(v) = \begin{cases} 0 & \text{if } v \in PIs \\ \max\limits_{p \in P(v)} \{\ell(p)\} + 1 & \text{otherwise.} \end{cases} \tag{2.2}$$

2.3.2 RDR Architecture

The RDR architecture [15] enables multi-cycle communication while considering interconnection delays in high-level synthesis. The chip is divided into several regular islands and registers are distributed onto each island. We can easily estimate the interconnection delays due to its regularity.

Figure 2.1 shows RDR architecture consisting of 2×3 islands. Every island has three elements: A set of FUs, which execute operations, and their inputs/outputs are connected to registers; A set of registers a storage unit for the island; Finite state machine (FSM) controls FUs, registers, and MUXs associated with them in the island.



(a) Scheduled DFG.

(b) Floorplan of FUs and interconnection between islands.

Figure 2.2: An example of multi-cycle data transfer between islands.

The size of an island is determined to complete its single-cycle operation and writing/reading registers in the same island within a single clock cycle. The islands are connected by the global interconnection to each other.

Every island is assumed to be square. Assume that the entire chip is divided into $N \times M$ islands. The island in n-th row and m-th column is denoted by $I(n, m)$ where $1 \leq n \leq N$ and $1 \leq m \leq M$. The data transfer time $D_c(i_1, i_2)$ from the island $i_1 = I(n_1, m_1)$ to the island $i_2 = I(n_2, m_2)$ is defined as follows [27, 57]:

$$D_c(i_1, i_2) = C_d \cdot (|n_1 - n_2| + |m_1 - m_2|)^2 \tag{2.3}$$

where C_d is the interconnection-delay coefficient.[3]

Assume that an FU fu_1 is placed in the island $i_1 = I(n_1, m_1)$ and we want to transfer the output data from the FU fu_1 to an FU fu_x placed in the island $i_x = I(n_x, m_x)$. We assume that the clock period is denoted by T_{clk} and $dsum(fu_1) \leq T_{clk}$.

In the case of $T_{clk} \geq D_c(i_1, i_x) + dsum(fu_1)$:

The operation in fu_1 and the data transfer to the island i_x are done in a single control step.

In the case of $T_{clk} < D_c(i_1, i_x) + dsum(fu_1)$:

After the operation in fu_1 finishes, the data is temporarily stored in a register in

[3]We determine that interconnection delays are in proportion to the square of the distance as in Eq. (2.3) just based on [27] and [57] using the Elmore delay model [23]. We can also assume that the interconnection delays are in proportion to the distance between RDR islands. Even in this case, a similar discussion can hold true.

the island i_1. After that, it is transferred to a register in the island i_x taking next $\lceil D_c(i_1, i_x)/T_{clk} \rceil$ steps as in the original RDR architecture [15]. Figure 2.2 shows an example of multi-cycle data transfer between islands. When we transfer data from register r_0 to register r_1 over two clock cycles, no registers are inserted in the global interconnection between registers. Register r_0 continues to output data in *(cs 2)* and *(cs 3)* and register r_1 receives data at *(cs 4)*.

If $dsum(fu_1) > T_{clk}$, a similar discussion can hold true.

An island has a capacity cost C, and an FU fu has the area cost c_{fu}. The sum of the area cost c_{fu} of all the FUs placed in an island must not exceed C.

2.3.3 Operation Chainings on RDR Architecture

In this subsection, we define operation chainings on RDR architecture considering interconnection delays. We assume that the number of operations to be chained is just two. Therefore, we read the data from a register, use two FUs, and write the data to a register in an operation chaining.

Assume that operations v_1 and v_2 are bound to FUs fu_1 and fu_2, respectively. The result produced by v_1 is used by v_2, i.e., v_2 is dependent on v_1. The number of cycles $cycle(fu_1)$ to execute fu_1 is defined by

$$cycle(fu_1) = \left\lceil \frac{dsum(fu_1)}{T_{clk}} \right\rceil. \tag{2.4}$$

The combined delay $dcon(fu_1, fu_2)$ for the two FUs, which is the sum of register read/write delay and delays for executing FUs fu_1 and fu_2, is expressed by

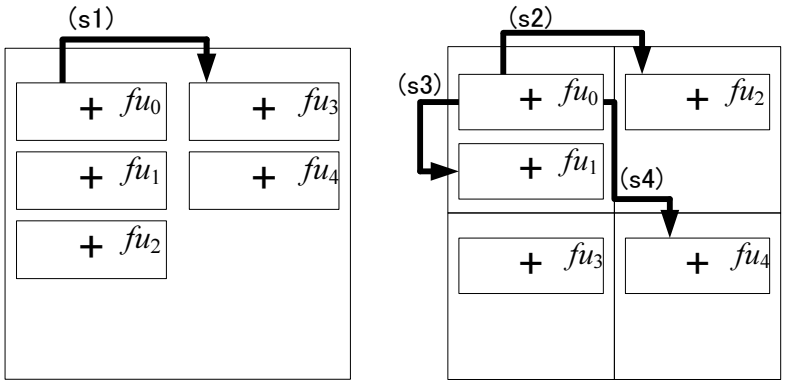
$$dcon(fu_1, fu_2) = d_{reg} + d_{fu_1} + d_{fu_2}. \tag{2.5}$$

Now, assume that the FUs fu_1 and fu_2 are placed in the islands i_1 and i_2, respectively. If the equation below is satisfied, the operations v_1 and v_2 can be chained.

$$dcon(fu_1, fu_2) + D(i_1, i_2) \leq T_{clk} \cdot cycle(fu_1) \tag{2.6}$$

Eq. (2.6) means that the operation v_2 is packed into the last control step executing the operation v_1 without violating data transfer time between v_1 and v_2. Let v_1 be an operation in the input DFG. Then we can define a set $CN(v_1) \subseteq S(v_1)$ of *chainable nodes to* v_1 by using Eq. (2.8) described later.

Figure 2.3 shows operation chainings on SR architecture versus those on RDR architecture. HLS algorithms targeting SR architectures do not usually consider the interconnection delays. Now, assume that the clock period T_{clk} is 3.0ns, the delay of every FU is 1.1ns, and the writing/reading delay of a register is 0.1ns. In SR architecture



(a) Operation chainings on an SR architecture.

(b) Operation chainings on an RDR architecture.

Figure 2.3: Operation chainings on SR architecture versus those on RDR architecture.

(Fig. 2.3(a)), fu_0 and fu_3 connected by the signal line (s1) can be chained, since the sum of the delays of the two FUs and a register calculated by Eq. (2.5), $(1.1 + 1.1 + 0.1)$, is smaller than T_{clk}. However, after the actual floorplanning is done and these two FUs are placed apart from each other, these two FUs cannot always be chained.

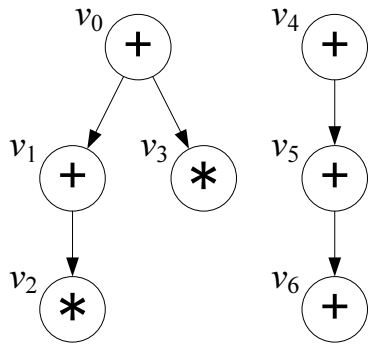
On the other hand, our proposed algorithm which targets RDR architecture considers the communication delay between the islands in HLS. Now, assume that the data transfer time between adjacent islands is 0.4ns and data transfer time between diagonally-placed islands is 1.6ns. In RDR architecture (Fig. 2.3(b)), fu_0 and fu_2 connected by the signal line (s2) can be chained, since the sum of the delays of the two FUs and a register, and the transfer time between islands, $(1.1 + 1.1 + 0.1 + 0.4)$, is smaller than T_{clk}. Similarly, fu_0 and fu_1 connected by the signal line (s3) can also be chained. However, we can know in HLS stage that fu_0 and fu_4 connected by the signal line (s4) cannot be chained on RDR architecture, whereas they can be considered to be chainable on SR architecture.

2.3.4 Problem Definition

Based on the discussion above, we define our HLS problem with operation chainings for RDR architecture as follows:

Definition 2.1. *For a given DFG $G = (V, E)$, the clock period T_{clk}, and the specifications of RDR architecture, our HLS problem with operation chainings is, to schedule and bind the input DFG and generate its floorplan on RDR architecture so as to minimize the latency while allowing the operation chainings consisting of two operations.* □

Example 2.1. *Figure 2.4 and Figure 2.5 show the input and the output of our HLS*

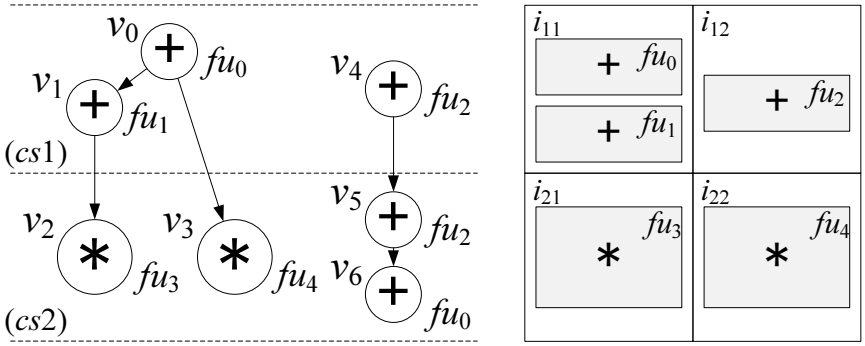


(a) DFG.

T_{clk}	3.0 ns
# of islands	2×2
# of FUs	Adder×3 (fu_0, fu_1, fu_2) Multiplier×2 (fu_3, fu_4)
Delay of an Adder	1.1 ns
Delay of a Multiplier	2.6 ns
Delay of a register	0.1 ns
Transfer time to adjacent islands (C_d)	0.4 ns
Transfer time to diagonally-placed islands	1.6 ns

(b) The clock period and the specifications of the RDR architecture.

Figure 2.4: Inputs to our HLS problem with operation chainings for RDR architecture.



(a) The scheduled and bound DFG.

(b) The floorplan result.

Figure 2.5: Outputs of our HLS problem with operation chainings for RDR architecture.

problem with operation chainings for RDR architecture, respectively. Figure 2.4(a) is the input DFG. Figure 2.4(b) shows the clock period and the specifications of RDR architecture. Figure 2.5(a) shows scheduled and bound DFG where the operations v_0 and v_1 are the chained nodes and executed within a single control step. v_5 and v_6 are also chained. Figure 2.5(b) shows the floorplan result. □

Generally, in HLS algorithms, synthesis is performed onto DFGs (which can be extended to CDFGs) [21]. In this dissertation, since the standard HLS benchmarks are provided as DFGs (or CDFGs) as in [25], we deal with DFGs (or CDFGs) in our HLS. However, an abstract behavioral description can be also represented as different DFGs (or CDFGs). In the future, we need to investigate to search the optimal solution by combining with the front-end and the optimization of the software-compiler. This is one

of the future works.

2.4 Proposed Algorithm

In this section, we propose our HLS algorithm with operation chainings for RDR architecture. We first analyze some conventional approaches, discuss the strategy, and then describe each step of the proposed algorithm.

2.4.1 Analysis of Conventional Approaches

In this subsection, we first analyze the conventional approaches, particularly the approaches in [15] and [74] and demonstrate that it is difficult to apply them directly to our HLS problem.

Firstly, the approach in [15] has been proposed for RDR architecture but does not accept any operation chainings, while it accepts multi-cycle operations. When an operation has a smaller delay than the given clock period, we may have an extra slack time in every clock cycle. We can reduce the latency furthermore if we utilize these slack times somehow.

Secondly, the approach in [74] has been proposed for conventional SR architecture and accepts operation chainings. It first enumerates chainable operations along every DFG critical path and then generates operation chainings based on them. However, it does not take into account interconnection delays explicitly when generating operation chainings since we do not have any information on interconnection delays.

When we apply conventional SR-based approaches with operation chainings to RDR architecture, we may face several problems as follows: Since conventional approaches do not take into account interconnection delays explicitly, the easiest way that we can apply them to RDR architecture is that we give a constant delay to each operation as an interconnection delay. We can consider the following two cases:

1. We give the maximum possible interconnection delay to each operation node.

2. We give a constant value such as an average interconnection delay value to each operation node.

In Case 1, for example, the maximum interconnection delay between RDR islands is estimated to be 23.04ns in the largest RDR architecture used in our experiments in Section 2.5. This value is sixteen times as long as the adder delay (which is just 1.44ns). It is extremely pessimistic to give this value to each operation and we cannot construct operation chainings efficiently. In Case 2, consider that we give an interconnection delay value smaller than the maximum interconnection delay. Even if we can construct

operation chainings in HLS stage, we cannot construct their operation chainings after actual floorplanning. We can conclude that neither the Case 1 nor Case 2 are practical.

Based on the discussions above, we have to solve the problems below in our proposed algorithm:

(i) Apply operation chainings to the critical paths explicitly including interconnection delays in RDR architecture.

(ii) Reduce the overall latency by reducing the critical path delays.

2.4.2 Strategy

Let us consider several strategies to reduce the latency by applying operation chainings to HLS for RDR architecture. Since MCAS [15] consists of scheduling, binding, allocation, and floorplan, we consider each of them.

Scheduling and binding are the most important steps when we apply operation chainings to HLS. We can reduce the latency by searching the chainable operation nodes, assigning those operation nodes to control steps (scheduling), and binding those operation nodes to different FUs (binding).

Allocation is also an important step because adding extra FUs may increase chainable nodes. However adding extra FUs may violate island capacity and we cannot complete an operation and writing/reading registers in each island within pre-determined clock cycles. In this chapter, we add no FUs and we consider HLS under given FUs.

As an initial floorplan, we can use placement generated by [15]. The result must be an optimal one in HLS without chainings. Since we use the same DFG even when we apply operation chainings to HLS, we do not expect to generate a better solution by changing the floorplan. On the contrary, changing the floorplan can affect overall scheduling and binding, which can lead to a worse result. We conclude that changing floorplan is not a good choice and thus we do not change the initial floorplan.

Based on the considerations above, our proposed algorithm first generates the placement of FUs by using MCAS ("scheduling-driven placement") [15] and then perform operation chainings in scheduling and binding.

To perform operation chainings in scheduling and binding, we can have two solutions below:

Solution 1: After preparing chaining nodes beforehand under several constraints, we schedule and bind these nodes.

Solution 2: We just enumerate *candidate* nodes for chaining at first. After that, we determine the actual chaining nodes.

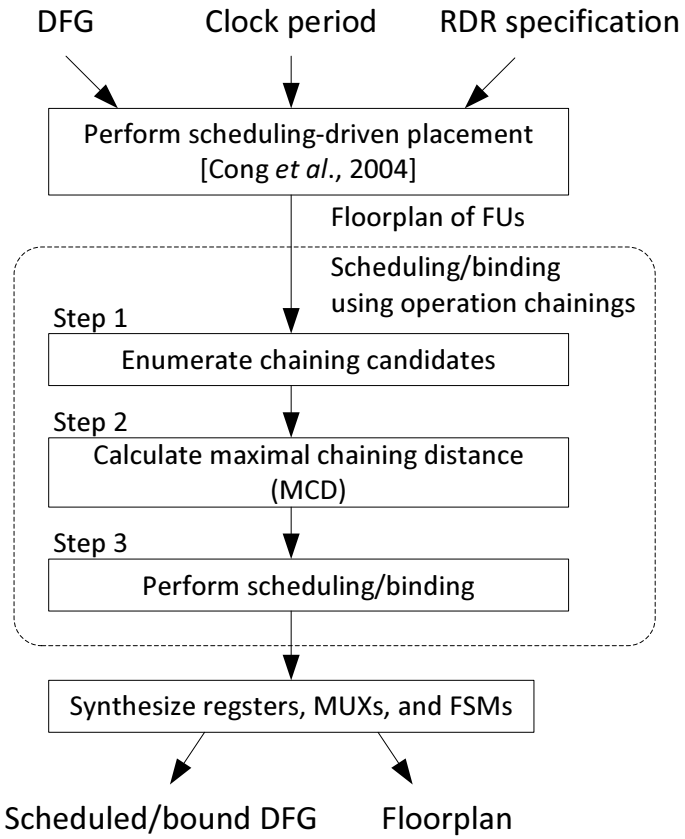


Figure 2.6: Synthesis flow of our algorithm.

Although RDR architecture has an advantage of estimating interconnection delays easily, we cannot estimate them before operation binding. This means that Solution 1 above is too difficult to employ. On the other hand, Solution 2 just enumerates as many candidate nodes as possible to be chained and, after that, we actually determine which nodes are chained through scheduling and binding. Solution 2 is a reasonable option to construct our HLS algorithm using operation chainings for RDR architecture.

Based on the discussion above, we first enumerate candidate operation nodes for chaining, and then we actually schedule and bind the input DFG using the candidate nodes.

2.4.3 Synthesis Flow

The synthesis flow of our algorithm is shown in Fig. 2.6. Initially, we determine on which island every FU is placed by using MCAS ("scheduling-driven placement") [15].

"Scheduling/binding using operation chainings" consists of three steps: In Step 1, we enumerate candidate nodes for operation chaining. In this step, we do not consider

data transfer time between the candidate nodes. In Step 2, we newly introduce *maximal chaining distance (MCD)* and calculate this value. By using MCD, our algorithm takes into account the data transfer time and enables multi-cycle communication even for operation chainings. We cannot know whether the FUs executing chained operations are placed in the same island or not in Step 1. Then Step 1 just enumerates as many candidate of chainings as possible without taking into account interconnection delays. After that, we give the MCD value to each enumerated candidate chaining in Step 2. In Step 3, we simultaneously schedule and bind the input DFG. In this step, we set a priority to each node and determine which candidate nodes actually construct operation chainings by using MCD. This step reduces the critical path delays by constructing chainings and assigning them to as early control steps as possible. List-scheduling is one of reliable approaches to realize this step. By effectively designing priorities in list-scheduling, we expect to reduce the latency of the given DFG. Overall, we can solve the problems i) and ii) enumerated in Section 2.4.1 through Step 1 to Step 3.

Our algorithm finally synthesizes registers, MUXs, and FSMs after completing scheduling/binding.

Since the core of our flow is "scheduling/binding using operation chainings", we describe it in the rest of this subsection.

Enumerate Candidates for Chaining (Step 1)

Step 1 enumerates the candidates for chaining. The key point of this step is that we do not determine the nodes for chaining directly. We allow that some nodes are included in two or more candidates for chaining and we actually determine the operation chainings in a later step.

For example, we enumerate $\langle v_4, v_5 \rangle$ and $\langle v_5, v_6 \rangle$ as candidates for chaining in Fig. 2.4(a). The node v_5 are included in both the candidates.

The candidates for chaining are enumerated as follows: We first calculate an *imaginary cycle* for each edge in a DFG $G(V, E)$. Let $e_{v_1,v_2} \in E$ be an edge in the DFG and v_1 and v_2 can be executed by FUs fu_1 and fu_2, respectively. For the edge e_{v_1,v_2}, an imaginary cycle $icycle(e_{v_1,v_2})$ is defined by

$$icycle(e_{v_1,v_2}) = \begin{cases} 0 & \text{if } dcon(fu_1, fu_2) \leq T_{clk} \cdot cycle(fu_1) \\ 1 & \text{otherwise.} \end{cases} \tag{2.7}$$

If the imaginary cycle is 0, it means that the sum of operation delays that the edge e connects does not exceed the control steps originally required for fu_1. If the imaginary cycle is 1, it means that the sum of operation delays that the edge e connects exceeds the control step originally required for fu_1.

After calculating imaginary cycles for all the edges, we determine whether the chainings can be constructed or not and enumerate candidates for chaining. Let $v \in V$ be a

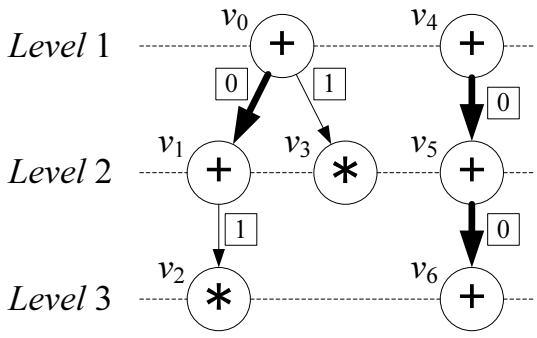


Figure 2.7: Enumeration of chaining candidates in Step 1.

node in a DFG where its level $\ell(v) = n$. Then a set $CN(v)$ of chainable nodes to v can be defined by

$$CN(v) = \left\{ u \in S(v) \mid \ell(u) = n + 1 \wedge icycle(e_{v,u}) = 0 \right\}. \tag{2.8}$$

Since the level of any node $u \in CN(v)$ is larger by one than that of v, there exists a direct edge from v to u and there are no paths from v to u via other nodes. $icycle(e_{v,u}) = 0$ means that the sum of the delays of the two operations that the edge $e_{v,u}$ connects does not exceed the control steps originally executing v. In this case, the nodes v and u can be chained unless they are placed apart from each other. In this sense, $CN(v)$ gives candidates for chaining for the node v.

Example 2.2. *Figure 2.7 shows an example of Step 1 when we give Fig. 2.4 as an input. Operation nodes in the same height connected by a dotted line have the same level, and the number written in the square beside the edge represents the imaginary cycle. For the operation v_0, we have $\ell(v_0) = 0$, $\ell(v_1) = 1$, $\ell(v_3) = 1$, $icycle(v_0, v_1) = 0$, and $icycle(v_0, v_3) = 1$. Then we have $CN(v_0) = \{v_1\}$. In the same way, we have $CN(v_1) = \emptyset$, $CN(v_2) = \emptyset$, $CN(v_3) = \emptyset$, $CN(v_4) = \{v_5\}$, $CN(v_5) = \{v_6\}$, and $CN(v_6) = \emptyset$.* □

Calculate Maximal Chaining Distance (Step 2)

Step 2 calculates a maximal allowable inter-island distance for each candidate for chaining obtained by Step 1. This is called maximal chaining distance (MCD). Let $i_1 = I(n_1, m_1)$ and $i_2 = I(n_2, m_2)$ be two islands on RDR architecture. The inter-island distance between them is give by $|n_1 - n_2| + |m_1 - m_2|$. Let v_1 and v_2 be two nodes in the input DFG where $v_2 \in CN(v_1)$. Let $d(v_1)$ and $d(v_2)$ be the FU delays to execute v_1 and v_2, respectively. Let $cycle(v_1)$ be the required cycle to execute v_1. Then MCD is defined by

$$MCD(v_1, v_2) = \left\lfloor \sqrt{\frac{T_{clk} \cdot cycle(v_1) - \left(d(v_1) + d(v_2) + d_{reg}\right)}{C_d}} \right\rfloor. \tag{2.9}$$

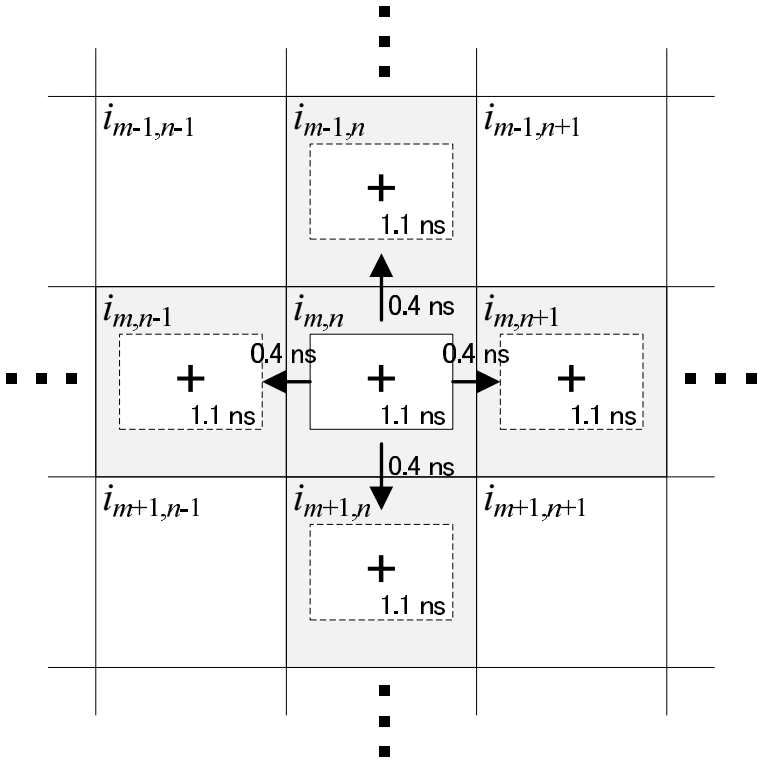


Figure 2.8: Calculating MCD in Step 2.

In Eq. (2.9), $[T_{clk} \cdot cycle(v_1) - (d(v_1) + d(v_2) + d_{reg})]$ is the difference between the product of the clock period and the required steps for executing v_1 ($T_{clk} \cdot cycle(v_1)$), and executing time of two operations ($d(v_1) + d(v_2) + d_{reg}$). It shows the slack time when executing v_1 and v_2 and it can be used for data transfer for them. Then the maximal inter-island distance can be calculated based on Eq. (2.3). The floor function should be applied since the inter-island distance is an integer. Overall, we can have Eq. (2.9).

Example 2.3. *Since $\langle v_0, v_1 \rangle$ in Fig. 2.4 is a chaining node candidate, we calculate $MCD(v_0, v_1)$ based on the values given by Fig. 2.4(b). v_0 and v_1 are the additions and the adder delay is 1.1ns. Since the clock period T_{clk} is 3.0ns, we can calculate* $MCD(v_0, v_1) = \left\lfloor \sqrt{\frac{3.0 \times 1 - (1.1 + 1.1 + 0.1)}{0.4}} \right\rfloor = 1$.

Assume that the FU fu_0 which executes v_0 is placed on the island $i_{m,n}$, as shown in Fig. 2.8. In this case, if the FU which executes v_1 is placed on the island whose inter-island distance from $i_{m,n}$ is up to $MCD(v_0, v_1)$, v_0 and v_1 can be chained. This means that, if v_1 is placed on $i_{m,n}$, $i_{m-1,n}$, $i_{m+1,n}$, $i_{m,n-1}$, or $i_{m,n+1}$, then v_0 and v_1 can be chained. □

Since MCD is the allowable inter-island distance between two operations, it can be applied to the adjacent two operations on the DFG but it cannot be simply applied to

three or more operations. Assume that we have three operations v_0, v_1, and v_2 connected serially and we simply apply our algorithm to them. If we calculate MCDs for the first two operations $\langle v_0, v_1 \rangle$ and the last two operations $\langle v_1, v_2 \rangle$ separately, the sum of $MCD(v_0, v_1)$ and $MCD(v_1, v_2)$ may exceed the maximal allowable inter-islands distance among the three operations and its scheduling result cannot satisfy the given clock cycle. In this situation, MCDs of the three operations do not make sense. In order to extend our operation chaining algorithm to three or more operations, we require extended MCDs for any three-connected operations or more. How to define these extended MCDs and how to apply them must be the challenging future works.

Perform Scheduling/binding (Step 3)

Step 3 performs scheduling and binding. We use list-scheduling as a scheduling algorithm. We have to execute the nodes on the critical path as early as possible to minimize the latency. According to [15], we calculate the priority based on the critical path length (CPL). When a node v is bound to an FU fu_i, the CPL from v to primary outputs is recursively calculated as follows:

$$cpl(v, fu_i) = dsum(fu_i) + \max_{v_k \in S(v)} \left\{ \min_{fu_l \in EFU(v_k)} \{D_c(i_i, i_l) + cpl(v_k, fu_l)\} \right\} \tag{2.10}$$

where i_i and i_l are the islands where the FUs fu_i and fu_l are placed, respectively. Based on the CPL, the priority of the operation node v is calculated by

$$priority(v) = \min_{fu_i \in EFU(v)} \{cpl(v, fu_i)\} . \tag{2.11}$$

List scheduling uses the priority queue PQ consisting of nodes which are ready to be scheduled. For every control step CS, we pick up the operation node v which is able to be scheduled and has the largest priority from PQ.

If the node v has no chainable nodes, i.e., $CN(v) = \emptyset$:

v is scheduled to the control step CS and bound to the FU fu_i with the smallest $cpl(v, fu_i)$ value. If there are no vacant FUs, v is scheduled to a later control step.

If the node v has chainable nodes, i.e., $CN(v) \neq \emptyset$:

v is scheduled and bound in the same way above.

After that, we try to chain the node $u \in CN(v)$ to v which has the maximum $priority(u)$ value. u is tried to be bound to an FU fu_j with the smallest $cpl(u, fu_j)$ value. If the inter-island distance between the island where fu_i is placed and the island where fu_j is placed does not exceed $MCD(v, u)$, u is actually bound to fu_j

and the operation chaining is constructed. If not, we try another FU to which u can be bound in the same way. If there are no other FUs to which u can be bound, we try another node in $CN(v)$ to be chained.

If we fail all the trials, only v is scheduled and bound and no chainings are constructed.

If there are no FUs to be bound or no operations are in the queue, we increase the control step count by one and repeat the same procedure. When we assign all the nodes in the input DFG to control steps and FUs, we finish scheduling/binding.

Example 2.4. *Assume that we have FU floorplan as shown in Fig. 2.5(b) for the input DFG of Fig. 2.4 by using [15]. We have* $CN(v_0) = \{v_1\}$, $CN(v_4) = \{v_5\}$, *and* $CN(v_5) = \{v_6\}$ *in Step 1. Other nodes have no chainable nodes. We also have* $MCD(v_0, v_1) = 1$, $MCD(v_4, v_5) = 1$, *and* $MCD(v_5, v_6) = 1$ *in Step 2.*

In Step 3, we first set the control step count as $CS = 1$ *and the priority queue PQ includes* v_0 *and* v_4. *Their priorities are calculated as* $priority(v_0) = 5.5$ *and* $priority(v_4) = 3.6$. *Since* v_0 *has the largest priority, it is firstly picked up from PQ and bound to the FU* fu_0. *Since* v_0 *has a chainable node* v_1, *we try to chain* v_1 *to* v_0. v_1 *can be successfully bound to the FU* fu_1 *within the maximal chaining distance (MCD). Then* v_0 *and* v_1 *can construct an operation chaining.*

Next, v_4 *is picked up from the priority queue and bound to the FU* fu_2. *Although* v_4 *has a chainable node* v_5, *it cannot be chained to* v_4 *since there are no available FUs. Since there is no operations in the queue now, we increase the control step count by one (*$CS = 2$*), we perform scheduling/binding in the same way. We finally obtain a scheduling/binding result as shown in Fig. 2.5(a).* □

2.5 Experimental Results

We have implemented our proposed algorithm in C++. We used CentOS 5.5 and AMD Quad-Core Opteron 2360 SE 2.5 GHz × 2 machine with 16GB memory. We applied our algorithm to six benchmark applications, PARKER (22 nodes including conditional branches), EWF (34 nodes), DCT (48 nodes), FIR filter (75 nodes), EWF3 (102 nodes), and COPY (378 nodes including conditional branches). FUs are assumed to have 16-bit width under the 90nm technology node. The capacity cost of an island is $C = 4$. The maximum size of an island is $240\mu\text{m} \times 240\mu\text{m}$ for COPY and it is $90\mu\text{m} \times 90\mu\text{m}$ for the other applications. The interconnection delay is in proportion to the square of the wire length and takes 1ns for $250\mu\text{m} \times 250\mu\text{m}$ [1]. The clock period is set to 4.0ns for COPY and 3.2ns for the other applications. The clock period here is defined by the smallest possible delay so that two consecutive additions on DFG can construct

an operation chaining when they are placed on two adjacent RDR islands. This is because we demonstrate that our algorithm efficiently constructs operation chainings on critical paths in a given DFG and then reduces its latency. We also optimize the best possible clock period for MCAS [15].[4] Controllers were synthesized with Synopsys Design Compiler. Design Compiler tries to minimize the controller area under the given clock period constraint. The capacity costs and delay of FUs with MUXs are shown in Table 2.1. We set the worst MUX delay to be 0.12ns as a constant and give it to each FU. We do not have multi-cycle operations in the experiments, even though we can handle them as described in Section 2.3 and Section 2.4. We place a memory unit in one of the islands.

We compared the three algorithms below to show the efficiency of our algorithm.

MCAS [15]: An original HLS algorithm for RDR architecture, which minimizes the latency under the given clock period and RDR specifications.

[74] + Proposed: Balanced chaining [74] is used to enumerate chaining nodes before our scheduling/binding is performed. After that, our our scheduling/binding is performed.

Proposed: Our proposed algorithm.

MCAS [15] is an HLS algorithm for RDR architecture but it does not deal with operation chainings. By comparing our algorithm to [15], we can demonstrate whether our algorithm can effectively apply operation chainings to critical paths including interconnection delays. The approach in [74] just enumerates chainable operations on critical paths in DFG not considering interconnection delays. By comparing our algorithm to " [74] + Proposed", we can demonstrate whether our algorithm can effectively enumerate chainable operation candidates on critical paths including interconnection delays.

The experimental results are shown in Table 2.2 and Table 2.3. In each benchmark application, the first row shows the results for MCAS with the best possible clock period and the other rows show the results for MCAS, " [74] + Proposed", and "Proposed" with the fixed clock period. The latency is calculated by multiplying T_{clk} and the control step counts. #chainings show the number of operation chainings, where each chaining is composed of two nodes. The maximum area among all the islands is shown in the "Max area" column, which is calculated by the sum of FU area, register area, and MUX area. Our algorithm reduces the latency compared to MCAS except for FIR. Our algorithm reduces the latency by up to 40.0% compared to MCAS, and by up to 25.0% compared to [74] for PARKER. Note that our algorithm does not reduce the latency for FIR but

[4] In MCAS, we tried several clock periods from 3.0ns to 5.0ns and picked up the one as the best possible clock period which gives the smallest latency for each benchmark application.

Table 2.1: The capacity costs and delay of FUs [1].

Functional unit	Capacity cost	Delay with MUXs [ns]
Adder	2	1.44
Subtractor	2	1.45
Multiplier	4	2.82
Right Shifter	2	0.67
Comparator	2	0.72
And	1	0.15
Memory unit	–	2.82
Register	–	0.11*

* Register delay does not include MUX delay.

does not increase it. Even when compared with MCAS with the best possible clock period, our algorithm reduces the latency by up to 36.0%.[5] Our algorithm reduces the number of registers by up to 33.3% compared to MCAS. Our algorithm also reduces the number of MUXs by up to 11.1% compared to MCAS. In general, we can reduce the number of registers when we apply operation chainings because the registers between the chained operations are not needed. The MUXs between the chained operations suppose to be reduced. However, we cannot say that we can always reduce the number of MUXs since it may increase by changing the FU bindings caused by operation chainings. In some applications, registers and MUXs cannot necessarily be reduced, since they have operation nodes which correlate to each other. In our algorithm, the maximum area among all the islands can be reduced, since the required control steps are reduced in most of the cases and then controller area can be reduced.

From the view point of enumeration of operation chainings, [74] mainly enumerates nodes on the critical path while not allowing duplicate nodes in chaining candidates. On the other hand, our algorithm effectively enumerate all the possible chaining candidates.

[5]In FIR, latency of MCAS with the best possible clock period is smaller than that of "Proposed." It is just because "Proposed" cannot construct operation chainings in this case. When we gave clock period of 3.06ns (which is the best possible clock period of MCAS) to "Proposed," our algorithm also realized the same latency as MCAS with the best possible clock period.

Table 2.2: Experimental results (1/2).

App.	#nodes	#islands	FUs	Algorithm	T_{clk} [ns]	Control steps	Latency [ns]	#chainings	#registers	#MUXs	Max area [μm^2]	CPU Time [sec]
PARKER	22	2×2	Add ×2, Sub ×2, Comp ×2	MCAS [15]	3.0	6	18.0 (93.8%)	–	9 (100.0%)	21 (100.0%)	2291	99.08
				MCAS [15]	3.2	6	19.2 (100.0%)	–	9 (100.0%)	21 (100.0%)	2255	95.25
				[74] + Proposed		6	19.2 (100.0%)	4	10 (111.1%)	21 (100.0%)	1695	95.90
				Ours		5	16.0 (83.3%)	5	7 (77.8%)	19 (90.5%)	1519	70.71
PARKER	22	2×3	Add ×4, Sub ×4, Comp ×4	MCAS [15]	3.0	5	15.0 (93.8%)	–	12 (100.0%)	36 (100.0%)	1757	143.49
				MCAS [15]	3.2	5	16.0 (100.0%)	–	12 (100.0%)	36 (100.0%)	1743	143.36
				[74] + Proposed		4	12.8 (80.0%)	5	10 (83.3%)	36 (100.0%)	1519	98.69
				Ours		3	9.6 (60.0%)	6	8 (66.7%)	32 (88.9%)	1407	74.07
EWF	34	2×3	Add ×6, Mul ×3	MCAS [15]	3.06	14	42.86 (95.7%)	–	11 (100.0%)	29 (100.0%)	5965	99.72
				MCAS [15]	3.2	14	44.8 (100.0%)	–	11 (100.0%)	29 (100.0%)	5434	122.99
				[74] + Proposed		12	38.4 (85.7%)	8	14 (127.3%)	29 (100.0%)	5691	148.31
				Ours		10	32.0 (71.4%)	11	10 (90.9%)	28 (96.6%)	5179	119.40
DCT	48	2×3	Add ×4, Mul ×4	MCAS [15]	3.0	9	27.0 (93.8%)	–	25 (104.2%)	41 (102.5%)	6516	149.06
				MCAS [15]	3.2	9	28.8 (100.0%)	–	24 (100.0%)	40 (100.0%)	6091	127.01
				[74] + Proposed		8	25.6 (88.9%)	5	21 (87.5%)	40 (100.0%)	6091	142.65
				Ours		8	25.6 (88.9%)	7	23 (95.8%)	39 (97.5%)	6315	142.07

Table 2.3: Experimental results (2/2).

App.	#nodes	#islands	FUs	Algorithm	T_{clk} [ns]	Control steps	Latency [ns]	#chainings	#registers	#MUXs	Max area [μm^2]	CPU Time [sec]
FIR	75	2×3	Add ×8, Mul ×2	MCAS [15]	3.06	21	64.26 (95.6%)	–	16 (100.0%)	38 (100.0%)	7164	100.95
				MCAS [15]	3.2	21	67.2 (100.0%)	–	16 (100.0%)	38 (100.0%)	6844	97.09
				[74] + Ours		21	67.2 (100.0%)	0	16 (100.0%)	38 (100.0%)	6844	95.81
				Ours		21	67.2 (100.0%)	0	16 (100.0%)	38 (100.0%)	6844	95.92
EWF3	102	2×2	Add ×4, Mul ×2	MCAS [15]	3.2	40	128.0 (100.0%)	–	15 (100.0%)	27 (100.0%)	6730	100.92
				MCAS [15]	3.2	40	128.0 (100.0%)	–	15 (100.0%)	27 (100.0%)	6730	100.92
				[74] + Ours		37	118.4 (92.5%)	18	14 (93.3%)	27 (100.0%)	6778	173.71
				Ours		30	96.0 (75.0%)	30	14 (93.3%)	26 (96.3%)	6554	149.19
EWF3	102	2×4	Add ×6, Mul ×5	MCAS [15]	3.06	40	122.4 (95.6%)	–	15 (88.2%)	37 (94.9%)	8084	106.46
				MCAS [15]	3.2	40	128.0 (100.0%)	–	17 (100.0%)	39 (100.0%)	7002	100.45
				[74] + Ours		33	105.6 (82.5%)	18	18 (105.9%)	39 (100.0%)	6330	146.66
				Ours		30	96.0 (75.0%)	31	16 (94.1%)	38 (97.4%)	6330	170.33
COPY	378	3×4	Add ×4, Sub ×2, Mul ×4, RShift ×4, Comp ×2, And ×2	MCAS [15]	4.0	85	340.0 (100.0%)	–	140 (100.0%)	178 (100.0%)	45608	708.39
				MCAS [15]	4.0	85	340.0 (100.0%)	–	140 (100.0%)	178 (100.0%)	45608	708.39
				[74] + Ours		83	332.0 (97.6%)	70	118 (84.3%)	156 (87.6%)	47768	669.63
				Ours		81	324.0 (95.3%)	103	134 (95.7%)	172 (96.6%)	44356	671.71

2.6 Conclusion

In this chapter, we proposed an HLS algorithm with operation chainings for RDR architecture which explicitly considers interconnection delays. Experimental results show that our proposed algorithm reduces the latency by up to 40.0% compared to [15], and by up to 25.0% compared to [74]. Our algorithm also reduces the number of registers and the number of multiplexers in some cases.

On the other hand, our algorithm does not reduce the latency for some applications. Solve this problem by enumerating two or more nodes as chainable nodes is one of the future works.

Chapter 3

A Floorplan-Aware High-Level Synthesis Algorithm with Multiple-Operation Chainings Based on Path Enumeration[1]

3.1 Introduction

In Chapter 2, a high-level synthesis algorithm using operation chainings is proposed. It reduces the overall latency targeting RDR architectures. However, it constructs operation chainings with only up to two operations. In this chapter, the algorithm proposed in Chapter 2 is extended to enumerate multiple-operation-chaining path candidates to generate operation chainings with multiple-operation and reduce more latency.

As process technologies advance, highly integrated circuits are strongly required and high-level synthesis (HLS) is becoming a very important design technique. HLS synthesizes register-transfer level circuits from abstract behavioral descriptions, while performing scheduling, allocation, and binding.

Highly integrated circuits cause interconnection delays to be relatively larger than gate delays, which has been a main concern even in HLS. To cope with this problem, regular-distributed-register (RDR) architecture has been introduced in [15]. RDR architecture is one of the distributed-register (DR) architectures in which registers are distributed over a chip, while registers are concentrated in conventional shared-register (SR) architectures. RDR architecture divides a chip area into several *islands* and enables multi-cycle communications on a chip. Since each island has the same size, we can easily estimate interconnection delays in HLS. An HLS algorithm for RDR architecture

[1]Technical contents in this chapter have been presented in the publications ⟨3⟩ and ⟨13⟩.

called MCAS was also proposed in [15].

Also we have to deal with operation delays in HLS since every operation has a different delay generally. For example, an adder typically has a smaller delay than a multiplier. *Operation chainings* can solve this problem where we pack several data-depending operations into packed control steps and reduce the overall latency.

In this chapter, we propose an HLS algorithm with multiple-operation chainings for RDR architecture. Our algorithm realizes a *multiple-operation chaining* by enumerating feasible chaining paths in K steps, where K is given as a *chaining search depth*. After enumerating chaining path candidates, our algorithm performs a scheduling/binding taking into account explicitly interconnection delays between RDR islands and thus realizes floorplan-driven multiple-operation chainings. Experimental results show that our algorithm reduces the latency by up to 30.4% compared to the conventional approaches.

The main contributions of this chapter are:

1. We realize *floorplan-driven multiple-operation chainings* in our HLS algorithm which explicitly takes into account interconnection delays between operations.

2. Experimental results successfully demonstrate that our HLS algorithm efficiently reduces the latency by 30.4%.

This chapter is organized as follows: Section 3.2 reviews related works; Section 3.3 describes our problem formulation; Section 3.4 proposes our HLS algorithm with multiple-operation chainings for RDR architecture; Section 3.5 shows experimental results; Section 3.6 gives several concluding remarks.

3.2 Related Works

Many HLS algorithms with operation chainings have been studied as in [40,43,52,63,67, 74]. All of these studies are, however, based on SR architecture and no HLS algorithms with operation chainings have been proposed for RDR architectures except for ours. We have proposed an HLS algorithm using operation chainings targeting RDR architectures in Chapter 2, where the maximum number of chainable operations is just two. How to realize a *multiple-operation chaining* composed of two or more operations is one of the key concerns.

3.3 Problem Formulation

In this section, we define control-data flow graphs and RDR architecture. Then we define multiple-operation chainings on RDR architectures. After that we define our HLS problem.

3.3.1 Control-Data-Flow Graphs

An input of HLS is a behavioral description represented by a control-data flow graph (CDFG). For simplicity, we use a DFG in this chapter to explain our algorithm but we can extend it to CDFG easily as in our experimental results.

A DFG is usually represented by a directed graph. Let $G = (V, E)$ be a DFG where V is a set of operation nodes and E is a set of edges between them.

Let d_{fu} be a delay of an FU fu and d_{reg} be the sum of reading and writing time of a register. We assume that an FU delay and a register delay include MUX delays.

For simplicity, each operation can be bound to one type of FUs, and an FU can execute only one type of operations. Therefore, each operation v has a unique delay $d(v)$. Let $EFU(v)$ be a set of FUs which can execute an operation v, given as an input.

We define a path on a DFG. A path from the node v_1 to the node v_n via the nodes $v_2, v_3, \cdots, v_{n-1}$ is denoted by $P = v_1\ v_2\ \cdots\ v_n$, where there exists an edge between v_i and v_{i+1} for $1 \leq i < n$. A path delay $d_{path}(P)$ is defined by

$$d_{path}(P) = d_{reg} + \sum_{v \in P} d(v) \tag{3.1}$$

where $P = v_1\ v_2\ \cdots\ v_n$.

3.3.2 RDR Architecture

The RDR architecture [15] enables multi-cycle communication while considering interconnection delays in HLS. The chip is divided into several regular islands and registers are distributed onto each island. We can easily estimate the interconnection delays for its regularity.

RDR architecture was shown in Fig. 2.1. Every island has three elements: A set of FUs; A set of registers is a storage unit for each island; Finite state machine (FSM) controls FUs, registers, and MUXs associated with them in the island.

The size of an island is determined to complete its single-cycle operation and writing/reading registers in the same island within a single clock cycle. The islands are connected by the global interconnection to each other. Every island is assumed to be square. Assume that the entire chip is divided into $N \times M$ islands. The island in n-th row and m-th column is denoted by $I(n, m)$ where $1 \leq n \leq N$ and $1 \leq m \leq M$. The data transfer time $D_c(i_1, i_2)$ from the island $i_1 = I(n_1, m_1)$ to the island $i_2 = I(n_2, m_2)$ is defined as follows [27, 57]:

$$D_c(i_1, i_2) = C_d \cdot (|n_1 - n_2| + |m_1 - m_2|)^2 \tag{3.2}$$

where C_d is the interconnection-delay coefficient.

We want to transfer the output data from an FU in an island i_1 to another FU in another island i_2. After the operation finishes in a control step in i_1, the data will be transferred to i_2 within the control step if the slack time is enough to transfer. Otherwise, the data is temporarily stored in a register in i_1 at the first control step and, after that, the data will be transferred to i_2 using the next control step or more.

An island has a capacity cost C, and an FU fu has the area cost c_{fu}. The sum of the area cost c_{fu} of all the FUs placed in an island must not exceed C.

3.3.3 Multiple-Operation Chainings on RDR Architecture

Unlike existing works, our multiple-operation chaining considers not only operation delays but *interconnection delays* among them. Since we use RDR architecture, we can effectively estimate them in HLS and solve this problem.

Let K be a *chaining search depth*. Let T_{clk} be a clock period. Then $CP(v)$ is defined by a set of paths in an input DFG whose start node is v and $d_{path}(P) \leq K \cdot T_{clk}$ for any path $P \in CP(v)$. Every path $P \in CP(v)$ is called a *chaining path* of v.

Assume that a node $v_1 \in V$ has only one chaining path, i.e., $CP(v_1) = \{P\}$, where $P = v_1 v_2 v_3 \cdots v_n$. Let P' be a sub-path in P whose start node is also v_1, i.e., $P' = v_1 v_2 v_3 \cdots v_m$ $(m \leq n)$. In this case, we can construct a *multiple-operation chaining* on RDR architecture for P' if it satisfies

$$d_{reg} + d(v_1) + \sum_{i=2}^{m} (d(v_i) + D_c(i_{i-1}, i_i)) \leq K \cdot T_{clk} \tag{3.3}$$

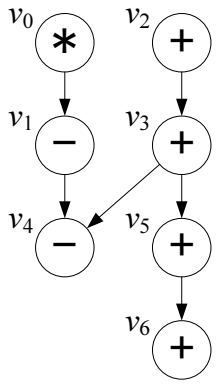
where we assume that the FU executing v_i is placed on the island i_i. Eq. (3.1) just shows the sum of operation delays but our multiple-operation chaining shown in Eq. (3.3) considers both operation delays and interconnection delays between them.

3.3.4 Problem Definition

Based on the discussion above, we define our HLS problem as follows:

Definition 3.1. *For a given DFG $G = (V, E)$, the clock period T_{clk}, a chaining search depth K, and the specifications of RDR architecture, our HLS problem with multiple-operation chainings is, to schedule and bind the input DFG and generate its floorplan on RDR architecture so as to minimize the latency while allowing the operation chainings in the chaining search depth.*

Example 3.1. *Figure 3.1 and Figure 3.2 show the input and the output of our HLS problem with multiple-operation chainings for RDR architecture, respectively. Figure 3.1(a) is the input DFG. Figure 3.1(b) shows the clock period and the specifications of RDR*

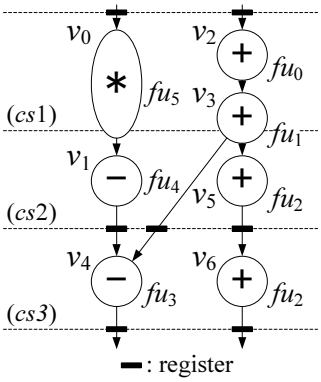


(a) DFG.

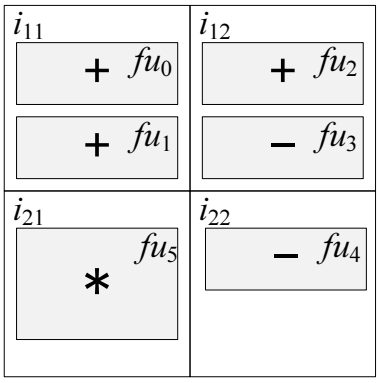
T_{clk}	3.0 ns
Chaining search depth (K)	2
# of islands	2×2
# of FUs	Adder×3 (fu_0, fu_1, fu_2) Subtractor×2 (fu_3, fu_4) Multiplier×1 (fu_5)
Delay of an adder / a subtractor	1.5 ns
Delay of a multiplier	2.8 ns
Delay of a register	0.1 ns
Transfer time to adjacent islands (C_d)	0.2 ns

(b) The clock period and the specifications of the RDR architecture.

Figure 3.1: Inputs to our HLS problem with multiple-operation chainings for RDR architecture.



(a) The scheduled/bound DFG.



(b) The floorplan result.

Figure 3.2: Outputs of our HLS problem with multiple-operation chainings for RDR architecture.

architecture. Figure 3.2(a) shows scheduled and bound DFG where the operations on the chaining paths $\{v_0, v_1\}$ *and* $\{v_2, v_3, v_5\}$ *are chained and executed in packed control steps. Figure 3.2(b) shows the floorplan result.*

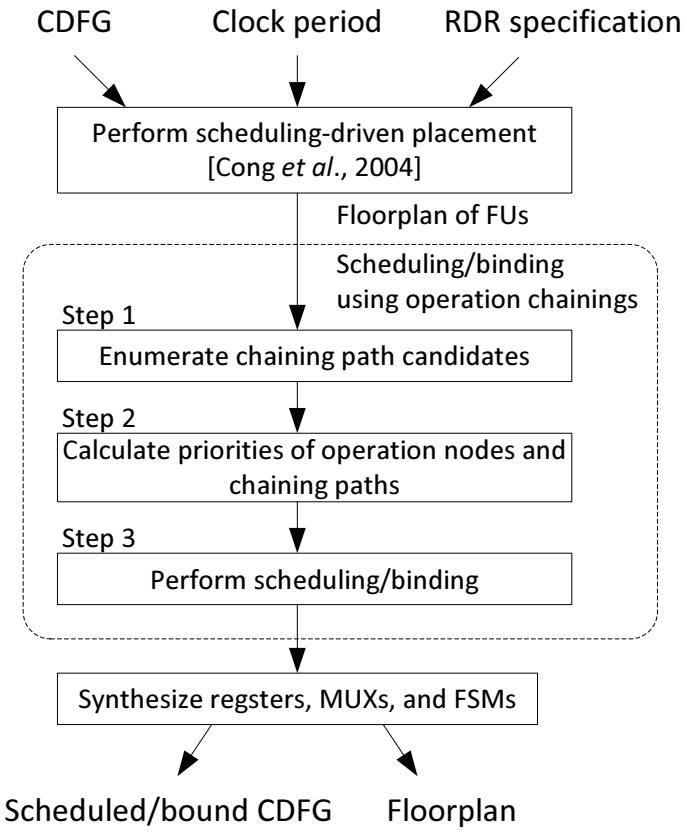


Figure 3.3: Synthesis flow of our algorithm.

3.4 Proposed Algorithm

In this section, we propose our HLS algorithm with multiple-operation chainings for RDR architecture.

Figure 3.3 shows our proposed synthesis flow. Initially, we determine which island every FU is placed in by using MCAS (scheduling-driven placement) [15] (Step 0).

The scheduling/binding using multiple-operation chainings consist of three steps: In Step 1, we enumerate candidate of chaining paths. In this step, we ignore data transfer time between operations and enumerate as many chaining paths as possible. In Step 2, we set the priorities for operation nodes and chaining paths. The priory of an operation node is calculated using cpl (critical path length) and the priority of a chaining path is calculated using the node priorities included in the path. We determine which chaining path is adopted as an actually chaining path based on the chaining path ordering obtained in this step. In Step 3, we simultaneously schedule and bind the input DFG based on the priorities calculated in Step 2. Our algorithm finally synthesizes registers, MUXs, and FSMs after completing scheduling/binding.

Since our synthesis flow is based on *enumeration* of candidate chaining paths and selects the best one among them, it can pack as many chainable nodes as possible and thus we can have optimal-latency scheduling/binding result considering RDR floorplanning.

Step 1: Enumerate chaining paths

Step 1 enumerates candidate of chaining paths for every node on DFG. Let $v \in V$ be a node in DFG. Then we visit every node from v in the depth-first-search manner and calculate the path delay. When the path delay from v to some node exceeds $K \cdot T_{clk}$, we stop searching into deeper nodes. Then we add the longest path searched so far whose path delay does not exceed $K \cdot T_{clk}$ into $CP(v)$ and try to search other paths from v in the same way.

Since we do not consider data transfer time between nodes at this step, we can enumerate the longest possible paths in this step.

Step 2: Calculate priorities of operation nodes and chaining paths

Operations on the critical path need to be executed earlier to minimize the overall latency. We calculate the priority of the nodes based on cpl (critical path length) as in [15]. The cpl from v to a primary output when a node v is bound to an FU fu_i is recursively calculated by

$$\begin{aligned} cpl(v, fu_i) = d_{reg} + d_{fu_i} + \\ \max_{v_k \in S(v)} \left\{ \min_{fu_j \in EFU(v_k)} \left\{ D_c(i_i, i_j) + cpl(v_k, fu_j) \right\} \right\} \end{aligned} \tag{3.4}$$

where we assume that fu_i and fu_j is placed on the islands i_i and i_j, respectively, and $S(v)$ shows a set of the successor nodes of v. The priority of a node v is calculated as

$$pr(v) = \min_{fu_i \in EFU(v)} \{cpl(v, fu_i)\} \tag{3.5}$$

Every path also has a priority. Each chaining path in $CP(v)$ is sorted in the following way, and the path priority is set in the sorted order. Assume that the number of chaining paths in $CP(v)$ is m. Each path of $CP(v)$ is represented by

$$\begin{cases} P_1 &= v_1^1 v_2^1 \cdots \\ P_2 &= v_1^2 v_2^2 \cdots \\ & \vdots \\ P_m &= v_1^m v_2^m \cdots \end{cases}$$

where the j-th node in a path P_i $(1 \le i \le m)$ is represented by v_j^i. First, we set $i = 1$ and sort the paths in the ascending order of $pr(v_1^i)$. If $pr(v_1^i)$ values are the same in several

paths, we increase i by one and repeat the same process until all the paths in $CP(v)$ are sorted.

Step 3: Perform scheduling/binding

Step 3 performs scheduling and binding. We use list-scheduling as a scheduling algorithm. In list-scheduling, a ready list RL is constructed as a priority queue which includes the nodes not assigned yet and ready to be scheduled. We use the resource pool $RP(cs) = \langle fu_1, fu_2, \cdots, fu_n \rangle$ which effectively shows a set of unused FUs at the control step cs even when we consider operation chainings.

Initially we set $cs = 1$ and, for each control step cs, we pick up the node v_1 with the largest priority from RL.

If the node v_1 has no chaining paths:

v_1 is scheduled to cs, and is bound to the FU fu having the smallest possible $cpl(v_1, fu)$ value considering the interconnection delays between v_1 and each of its parent nodes. If v_1 is successfully bound to fu, it is eliminated from $RP(cs)$. If we fail to bind v_1 to any FU at cs, v_1 is pushed back into RL and will be assigned in a later control step.

If the node v_1 has chaining paths:

v_1 is scheduled and bound in the same way above.

Let $P \in CP(v_1)$ be a chaining path with the largest path priority. Let v_2 be the second node in the path P. If all the conditions 1.–3. below are satisfied, v_2 is scheduled to cs and bound to the FU fu.

1. All parent nodes of v_2 are already scheduled.
2. There exists the FU fu at the control step cs which can execute v_2.
3. Except for v_1, all the input data to v_2 are ready to use for fu at cs considering interconnection delays between v_2 and each of its parent nodes.

If we fail to schedule/bind v_2, we stop constructing operation chainings for P. If we successfully schedule/bind v_2, fu is eliminated from $RP(i)$ for $cs \leq i \leq (cs + k)$ where v_1, v_2, and their interconnection delay between them occupies the control steps from cs to $(cs + k)$.

We repeat the same process for the remaining nodes in P until either condition below becomes true.

- We reach the end of the chaining path P.

Table 3.1: The capacity costs and delay of FUs [1].

Functional unit	Capacity cost	Delay [ns]
Adder	1	1.44
Subtractor	1	1.45
Multiplier	2	2.82
Comparator	1	0.72
Memory unit	–	2.82
Register	–	0.11

- The number of occupied control steps to construct chaining nodes exceeds K.

If we cannot construct any node chainings in P, we try a chaining path having the second largest priority in $CP(v_1)$ and we continue this process.

If there are no FUs to be bound or no operations are in the queue RL, we increase the control step count by one and repeat the same process.

3.5 Experimental Results

We have implemented our proposed algorithm in C++. We used CentOS 5.5 OS and AMD Quad-Core Opteron 2360 SE 2.5 GHz × 2 machine with 16GB memory. We applied our algorithm to four benchmark applications, PARKER (22 nodes including conditional branches), EWF (34 nodes), FIR filter (75 nodes), and EWF3 (102 nodes). FUs are assumed to have 16-bit width under the 90nm technology node. The clock period is set to 3.0ns. The capacity cost of an island is $C = 2$. The maximum size of an island is $90\mu\text{m} \times 90\mu\text{m}$. The interconnection delay is in proportion to the square of the wire length and takes 1ns for $250\mu\text{m} \times 250\mu\text{m}$ [1]. Controllers were synthesized with Synopsys Design Compiler. The capacity costs and delay of FUs are shown in Table 3.1. We place a memory unit in one of the islands.

We compared three algorithms below to show the efficiency of our algorithm.

MCAS [15]: We used the original HLS algorithm for RDR architecture without operation chainings.

HLS with chainings of up to two operation nodes (Chapter 2): We used the HLS algorithm for RDR architecture with operation chainings of up to two nodes proposed in Chapter 2.

Ours: We used our proposed algorithm with multiple-operation chainings. We set the chaining search depth $K \in \{1, 2\}$.

The experimental results are shown in Table 3.2. For PARKER, EWF, and EWF3, Ours ($K = 2$) realizes the smallest latency compared to the other algorithms. For FIR, Ours ($K = 1$) realizes the smallest latency.

Our algorithm reduces the latency by up to 30.4% compared to the algorithm without chainings [15], and reduces the latency by up to 24.4% compared to the algorithm with chainings of up to two nodes (Chapter 2) for EWF3.

Since our proposed algorithm is based on packing as many chainable nodes as possible in a give search depth, it is effective to the applications where many nodes are serially connected such as in EWF and EWF3.

Table 3.2: Experimental results.

App. #nodes	#islands	FUs	T_{clk} [ns]	Algorithm	Control steps	Latency [ns]	#registers	#MUXs	Max island area [μm^2]	CPU time [sec]
PARKER 22	2×3	Add×4, Sub×4, Comp×4	3.0	[15]	7	21.0 (1.00)	13 (1.00)	37 (1.00)	1756	139.45
				Ch. 2	5	15.0 (0.71)	12 (0.92)	36 (0.97)	1536	133.37
				Ours (K = 1)	7	21.0 (1.00)	11 (0.85)	35 (0.95)	2078	133.70
				Ours (K = 2)	5	15.0 (0.71)	10 (0.77)	34 (0.92)	1494	113.38
EWF 34	2×3	Add×6, Mul×3	3.0	[15]	17	51.0 (1.00)	13 (1.00)	31 (1.00)	6050	93.94
				Ch. 2	15	45.0 (0.88)	14 (1.08)	32 (1.03)	5662	137.73
				Ours (K = 1)	16	48.0 (0.94)	15 (1.15)	33 (1.07)	5717	141.10
				Ours (K = 2)	13	39.0 (0.77)	14 (1.08)	32 (1.03)	5113	120.29
FIR 75	2×3	Add×8, Mul×2	3.0	[15]	23	69.0 (1.00)	17 (1.00)	39 (1.00)	8060	74.71
				Ch. 2	23	69.0 (1.00)	19 (1.12)	41 (1.05)	8060	95.92
				Ours (K = 1)	22	66.0 (0.96)	18 (1.06)	40 (1.03)	7876	94.33
				Ours (K = 2)	33	99.0 (1.44)	18 (1.06)	40 (1.03)	8091	94.27
EWF3 102	3×3	Add×6, Mul×6	3.0	[15]	49	147.0 (1.00)	17 (1.00)	41 (1.00)	> 8100	103.26
				Ch. 2	45	135.0 (0.92)	18 (1.06)	42 (1.02)	8011	145.54
				Ours (K = 1)	48	144.0 (0.98)	18 (1.06)	42 (1.02)	6973	142.77
				Ours (K = 2)	34	102.0 (0.69)	22 (1.29)	46 (1.12)	6968	175.40

3.6 Conclusion

In this chapter, we proposed a floorplan-driven HLS algorithm with multiple-operation chainings for RDR architecture by using enumeration of chaining paths. Chaining paths of multiple-operation are constructed with a chaining search depth K and the optimal ones are selected based on path priorities. Experimental results show that our algorithm reduces the latency by up to 30.4% compared to the algorithm without chainings, and reduces the latency by up to 24.4% compared to the algorithm with chainings of up to two nodes proposed in Chapter 2.

Our algorithm may not by efficient for some applications, since it realizes operation chaining as deep as possible in a greedy way. One of the future works is to solve this problem by finding out the optimal size of chaining nodes.

Chapter 4

A Floorplan-Driven Bitwidth-Aware High-Level Synthesis Algorithm Using Operation Chainings[1]

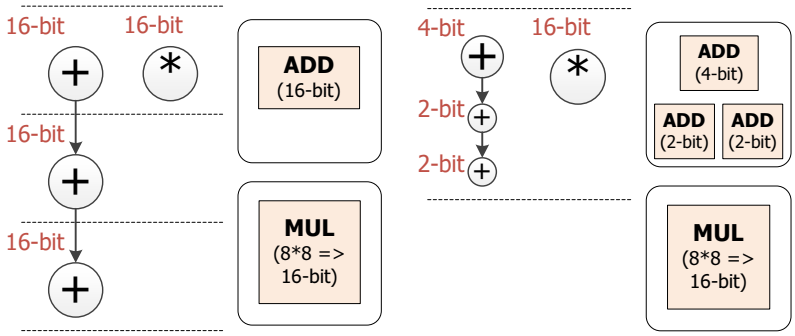
4.1 Introduction

In Chapter 2, a high-level synthesis algorithm using operation chainings is proposed. In Chapter 3, it is extended to deal with multiple-operation chainings. However, the algorithm proposed in Chapter 3 is only effective to the limited benchmarks. In this chapter, bitwidth-based optimization is used towards more performance optimization.

Thanks to the scaling of process technology, it comes to be able to produce highly integrated circuits, and larger applications can be implemented on VLSI SoCs. Hardware implementations for large applications in short time-to-market are strongly required. High-level synthesis (HLS) which synthesize from abstract application descriptions such as C, C++, Java, and Python into register transfer level (RTL) descriptions with low time cost, becomes a more and more essential and promising EDA technique nowadays.

A main concern in HLS is *interconnection delays*. In highly integrated SoCs, interconnection delays are relatively larger compared to logic delays. Adding the maximum possible interconnection delay to the clock cycle is a naïve way to solve the problem. However, in large circuits, the maximum interconnection delay is estimated extremely larger than logic delays and it worsens the overall performance, thus such approach is not practical. Taking interconnection delays into account in HLS is an effective solution to the problem by using distributed-register and -controller (DR) architecture, while registers and controllers are concentrated in conventional shared-register and -controller (SR) architectures. DR architecture is divided into clusters. Registers, FUs

[1]Technical contents in this chapter have been presented in the publications ⟨1⟩ and ⟨8⟩.



(a) Uniform bitwidth without operation chainings.

(b) Varied (non-uniform) bitwidths with an operation chaining.

Figure 4.1: Schedulings of uniform bitwidth vs. non-uniform bitwidth with operation chaining. The latency of scheduling (b) is lower than that of scheduling (a).

(Functional Units), and controllers are placed distributed onto the clusters, and we can leave the interconnection delays between registers and FUs being much smaller than SR architectures. Many floorplan-driven HLS algorithms targeting DR architectures which floorplan the modules layouts have been studied to take interconnection delays into account as in [2, 3, 15, 18, 28, 36, 45]. In some studies, they call such clusters on the DR architectures islands.

When we design hardwares, it is profitable to consider *data bitwidth*. According to an investigation in [55], on average 40% or more bits in operations are not used in DSP applications written in C language. Contrasting with a software design, we can design an arbitrary bitwidth for operations and variables on a hardware design and reduce the useless bits to obtain a circuits with less delays and areas. Bitwidth-aware HLS algorithms have been studied as in [18,20,35]. Proposed methods in [18,20,35] are HLS algorithms with bitwidth considerations, but these methods are targeting general SR architectures and they ignore interconnection delays.

Basically in HLS scheduling, each operation is assigned to a single clock cycle. However, every operation or FU has a different delay. For example, an adder usually has a smaller delay than a multiplier. We also have to deal with these operation delays in HLS with *operation chainings*. Operation chainings pack several data-depending operations into packed control steps and we can reduce the overall latency. Many HLS algorithms with operation chainings have been studied as in [13,18,54,56,75], but most of them are approaches targeting conventional SR architectures.

Since FUs with smaller number of bitwidth have smaller logic delays, we can effectively design operation chainings to reduce the overall latency as shown in Fig. 4.1.

Coussy *et al*. proposed a bitwidth-aware HLS algorithm using operation chainings in [20], however, selecting operation chainings in scheduling and binding is based on pattern matching of user-specified chaining patterns, thus it does not realize operation chainings automatically. Cong *et al*. proposed in [17] a bitwidth-aware HLS algorithm considering interconnection delays targeting RDR architectures [15], however, it focuses on the area minimization and does not realize operation chainings since the operation delays are assumed to be the same in every bitwidth. The allocation and floorplan result in [17] is the same as [15], since the allocation and floorplan result in [15] is optimized for the uniform bitwidth. [17] does not change them specified for the bitwidth consideration.

Cong *et al*. proposed RDR architecture for FPGAs called DRFM architecture in [18], in which registers are implemented as register files in FPGAs. [18] discusses a way of applying operation chainings in their HLS algorithm for DRFM architecture. However, DRFM architecture model does not calculate interconnection delays explicitly and an operation chaining is executed only in a single island. Furthermore, it does not evaluate the effects of their operation chainings. HLS algorithms with operation chainings targeting RDR architectures are proposed in Chapter 2 and Chapter 3. They realize operation chainings by taking inter-island distances into account and reduce the overall latency, however they do not consider operation bitwidths.

In this chapter, we propose a bitwidth-aware HLS algorithm using operation chainings to synthesize high-performance circuits targeting *RDR architectures*. RDR architecture is divided into *islands* on which registers and controllers are placed. Each *island* assumes to be the same size just like [11, 12, 15, 16, 18, 26, 28, 29] for simplicity, and we can easily estimate interconnection delays between islands on RDR architectures for its uniformity. Since RDR architectures have uniformly sized islands, there often exist some *vacant islands*. Our proposed algorithm takes the interconnection delays into account and utilizes *vacant islands* to realize operation chainings considering operation delays which depend on bitwidths, simultaneously.

The remainder of this chapter is organized as follows: Section 4.2 demonstrates our motivating examples and shows that conventional approaches do not produce good results; Section 4.3 describes formulation of RDR architectures and our problem definition; Section 4.4 proposes our bitwidth-aware HLS algorithm with operation chainings for RDR architectures; Section 4.5 shows experimental results; Section 4.6 gives several concluding remarks.

The main contributions of this chapter are:

1. We realize *floorplan-driven bitwidth-aware operation chainings* in our HLS algorithm which explicitly takes into account interconnection delays between operations targeting RDR architectures.

2. Our HLS algorithm utilizes *vacant islands* in RDR architectures by eliminating

unnecessary bitwidths and adding efficient extra FUs, and realizes operation chainings.

3. Experimental results successfully demonstrate that our HLS algorithm efficiently reduces the latency by up to 47% compared to the conventional approach without increasing the total area.

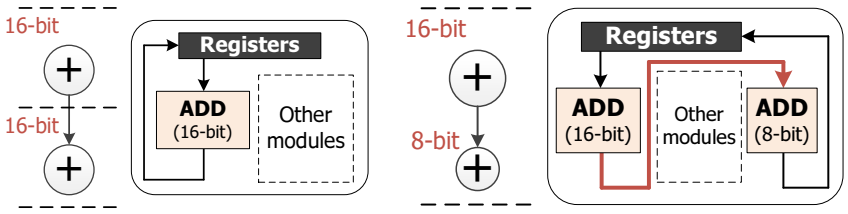
4.2 Motivating Examples

In previous works, there have been proposed methods (1) realizing bitwidth-aware operation chainings without considering module floorplans, (2) realizing module floorplan-based operation chainings without considering operation bitwidths, as described in Section 4.1. In this section, we demonstrate that both of above do not generate practical solutions through motivating examples.

4.2.1 Example 1: Approaches with vs. without Interconnection Delays Consideration

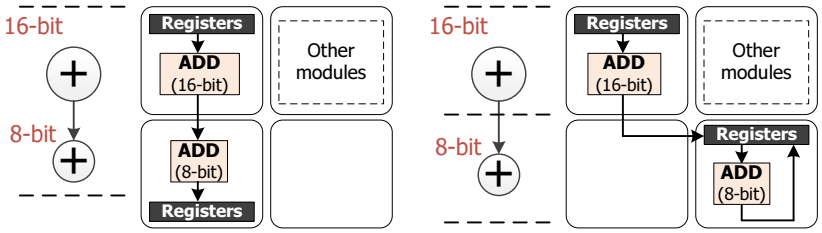
Figure 4.2 shows a motivating example to demonstrate that conventional HLS methods without interconnection delays considerations are not practical. Now, there are two consecutive adder operations and we are trying to assign these operations and perform physical floorplan. Let clock period be 3.0ns, and logic delays of 16-bit adder and 8-bit adder be 1.0ns and 0.5ns, respectively. Register delays are ignored in this example.[2] In Fig. 4.2(a), bitwidth of both operations are 16-bit and no operation chainings are realized on the SR architecture. Let us consider the operation bitwidth as in Fig. 4.2(b). In Fig. 4.2(b), bitwidth of the second adder operation is 8-bit and a operation chaining is realized by method [20] based on an SR architecture, but interconnection delay is too large to meet the clock period constraint. Typical HLS algorithm succeeds to schedule an operation chaining since just by adding logic delays of two operations and get the result is smaller than the clock period. However after physical placement, interconnection delays between FUs which two operations execute may be too large and violate the clock constraint. Figure 4.2(c) and Figure 4.2(d) show a realized practical operation chaining due to smaller interconnection delay on the RDR architecture and a solution of no operation chaining are realized due to larger interconnection delay on the RDR architecture, respectively. We need to get solutions like Fig. 4.2(c) and Fig. 4.2(d) rather than Fig. 4.2(b).

[2]Register delays are ignored just for the examples in this section, we calculate them in our proposed algorithm.



(a) Bitwidth of both operations are 16-bit and no operation chainings realized on an SR architecture.

(b) Bitwidth of the second operation is 8-bit and a operation chaining is realized by [20] based on an SR architecture, but interconnection delay is too large to meet the clock period constraint.



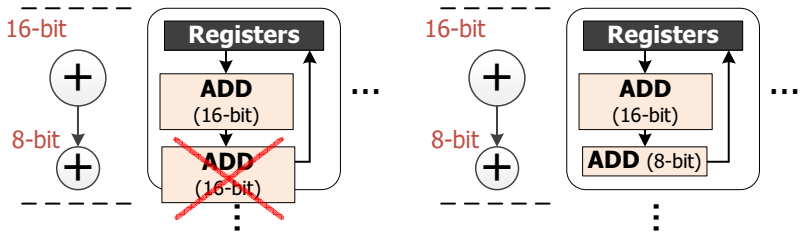
(c) Practical operation chaining is realized due to smaller interconnection delay on an RDR architecture.

(d) No operation chaining are realized due to larger interconnection delay on an RDR architecture.

Figure 4.2: A motivating example to demonstrate the operation chaining results without vs. with interconnection delays consideration.

4.2.2 Example 2: Approaches with vs. without Bitwidth Consideration

Figure 4.3 shows a motivating example to demonstrate that HLS methods without operation bitwidth considerations are not good enough. In general, an FU with smaller bitwidth has smaller delay/area. As described in Section 4.1, RDR architectures tend to have vacant spaces in islands for its regularity. We can place FU(s) to utilize them by considering bitwidth. Now, there are two consecutive adder operations and we are trying to schedule these operations as an operation chaining. Let clock period be 3.0ns, and logic delays/area of 16-bit adder and 8-bit adder be 1.0ns/160μm^2 and 0.5ns/80μm^2, respectively. We assume an island have a 250μm^2 capacity area, i.e., the sum of area of FUs placed on the island must not exceed 250μm^2. In this example, the interconnection



(a) Both operations are tried to be executed by 16-bit adders like in Chapter 2, but it fails due to short of vacant area.

(b) Practical result due to the second operation is executed by 8-bit adders.

Figure 4.3: A motivating example to demonstrate the operation chaining results without vs. with bitwidth of FUs consideration.

delay between two adders is considered as zero because they are placed in the same island, these two examples do not violate the timing. However, due to short of vacant area, we can obtain solution in Fig. 4.3(b) but cannot do in Fig. 4.3(a). We need to get solutions like Fig. 4.3(b) rather than Fig. 4.3(a).

4.3 Problem Formulation

In this section, we define control-data flow graphs and functional units. Then we define RDR architectures and operation chainings on RDR architectures. After that we define our HLS problem.

4.3.1 Control-Data-Flow Graphs and Functional Units with Bitwidth

An input to HLS is a behavioral description represented by a control-data flow graph (CDFG). For simplicity, we use a DFG in this chapter to explain our algorithm but we can extend it to CDFG.

A DFG is usually represented by a directed graph. Let $G = (V, B, E)$ be a DFG where V is a set of operation nodes, B is a set of bitwidth for all operations, and E is a set of edges between operation nodes. We can obtain bitwidth analysis result for each operation which satisfy the overall calculation accuracy or user-constraints by using bitwidth analysis methods like [33, 35, 55]. Therefore, we assume DFG and bitwidth for operations are given as DFG annotated with bitwidths in this chapter. Each operation $v_i \in V$ has its operation bitwidth denoted by $b_{op}(v_i)$. Similarly, each FU fu_i has its bitwidth $b_{fu}(fu_i)$. An operation v_i can be executed on FUs fu_j which satisfies

$b_{op}(v_i) \leq b_{fu}(fu_j)$. The operations which have no predecessors are called primary inputs (PIs) and a set of them is denoted by *PIs*. The operations which have no successors are called primary outputs (POs) and a set of them is denoted by *POs*. Let $EFU(v)$ be a set of FUs which can execute an operation v.

Let d_{fu} be a logic delay of an FU fu and d_{reg} be the sum of reading and writing time of a register. A delay of an FU fu depends on the FU bitwidths while that of a register does not. We assume that an FU delay and a register delay include MUX delays. Let $d_{single}(fu)$ be the sum of the delays of reading data from a register, logic delay on an FU fu, and writing data to a register, and it is calculated by:

$$d_{single}(fu) = d_{reg} + d_f(fu). \tag{4.1}$$

4.3.2 RDR Architecture

The RDR architecture is divided into several regularly sized islands and distributes FUs, local registers, and controllers onto the islands. We can leave the interconnection delays between local registers and FUs being much smaller than non-RDR architectures. Every island in RDR architecture assumes to be the same size just like [11, 12, 15, 16, 18, 26, 28, 29] for simplicity. We can easily estimate the interconnection delays between islands for its uniformly.

An example of RDR architecture model was shown in Fig. 2.1. Every island has three elements: A set of FUs; Registers as storage units for each island; Controller (FSM) which holds the current state and generates control (selection) signals to MUXs associated registers and FUs.

The islands are connected by the global interconnections to each other. Every island is assumed to be square. Assume that the entire architecture is divided into $N \times M$ islands. The island in n-th row and m-th column is denoted by $I(n, m)$ where $1 \leq n \leq N$ and $1 \leq m \leq M$. The data transfer time $D_t(i_1, i_2)$ from the island $i_1 = I(n_1, m_1)$ to the island $i_2 = I(n_2, m_2)$ is defined follows like in Chapter 2 and Chapter 3:

$$D_t(i_1, i_2) = C_d \cdot (|n_1 - n_2| + |m_1 - m_2|) \tag{4.2}$$

where C_d is the interconnection-delay coefficient.[3]

We want to transfer the output data from an FU in an island i_1 to another FU in another island i_2. After the operation finishes in a control step in i_1, the data will be transferred

[3]Interconnection delay is in proportion to the square of the distance in Chapter 2 and Chapter 3 based on Elmore delay model. However we assume that the interconnection delay is in proportion to the distance in this chapter because we can treat that the delay is in proportion to the distance by inserting some buffers onto the wires [14]. In our experiments in Section 4.5, we modify the delay estimation described in Chapter 2 and Chapter 3 to the model where the interconnection delay is in proportion to the distance. More details are described in Section 4.5.5.

to i_2 within the control step if the slack time is enough to transfer. Otherwise, the data is temporarily stored in a register in i_1 at the same control step as the FU execution, then the data will be transferred to i_2 using the next control step or more.

Every island has an area capacity A, and an FU fu has the area cost a_{fu}. The sum of the area cost a_{fu} of all the FUs placed in an island i is denoted by A_i, and it is constrained not to exceed A, i.e., $A_i \leq A(\forall i)$. The vacant area of an island i is calculated by $A - A_i$. When there exists vacant area in island i, we can append additional extra FUs whose area cost does not exceed $A - A_i$.

4.3.3 Operation Chainings on RDR Architecture

Unlike existing works, our operation chaining considers not only operation logic delays but interconnection delays. Since we use RDR architecture, we can adequately estimate them in HLS.

Let T_{clk} be a clock period. Assume that a path p in DFG G consists of n operations $v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow \cdots \rightarrow v_n$, and operation $v_i \in p$ is bound to FU fu_j, path execution delay $d_{path}(p)$ is calculated by:

$$d_{path}(p) = d_{reg} + \sum_{v_i \in p} \left\{ d_f(fu_j) + D_t(i_j, i_{j+1}) \right\}. \tag{4.3}$$

where we assume fu_j is placed in island i_j. The result of the last operation is transferred to an island i_{n+1}.

Let $PATH(v, u)$ be a set of all paths from $v \in V'$ to $u \in V'$ in a subgraph $G' = (V', B', E') \subseteq G$, where v is precedent of u in a topological order. Then, operations in G' are realized as an operation chaining and executed in a single control step, if it satisfies

$$\max_{p \in PATH(i,o)} \left\{ d_{path}(p) \right\} \leq T_{clk} \tag{4.4}$$

where $i \in PIs(G')$ and $o \in POs(G')$.

4.3.4 Problem Definition

We define our HLS problem as follows:

Definition 4.1. *For a given DFG with operation bitwidths $G = (V, B, E)$, the clock period T_{clk}, initial FU allocation, and the specifications of RDR architecture, our bitwidth-aware HLS problem with operation chainings is, to schedule and bind the input DFG and generate its floorplan on RDR architecture so as to minimize the latency while allowing the operation chainings and adding extra FUs into vacant area.* □

Example 4.1. *Figure 4.4 and Figure 4.5 show the input and the output of our bitwidth-aware HLS problem instance with operation chainings for RDR architecture, respectively. Figure 4.4(a) is the input DFG with bitwidth information. Figure 4.4(b) shows the clock period and the specifications of RDR architecture. Figure 4.5(a) shows scheduled and bound DFG where the operations* $\{v_1, v_2, v_4\}$ *are realized as an operation chaining and executed in packed single control step. Figure 4.5(b) shows the FU floorplan result. FU* fu_4 *(4-bit adder) is an additional extra FU. The chaining between operation* v_1 *and* v_4 *is completed in a single island* $I(1,1)$*, while the chaining between operation* v_1 *and* v_2 *is realized across the islands* $I(1,1)$ *and* $I(1,2)$*. The outputs of operation* v_2 *and* v_4 *are temporarily stored in islands* $I(1,2)$ *and* $I(1,1)$*, and transferred to islands* $I(2,2)$ *and* $I(2,1)$ *in the second control step, respectively. Figure 4.5(c) shows the FU characterization. The sum of FUs area in each island does not exceed 240.* □

4.4 Proposed Algorithm

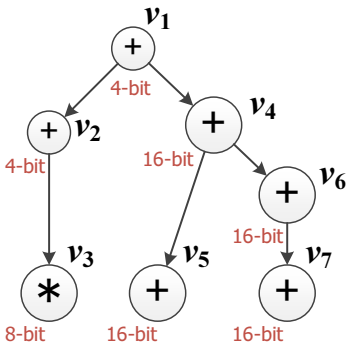
In this section, we propose a bitwidth-aware HLS algorithm with operation chainings for RDR architectures. Firstly, our FU characterization method and some strategies of our synthesis flow are discussed, and then our proposed algorithm is described.

4.4.1 FU Characterization

HLS needs to know delay and area information of FUs, and it performs characterization to generate FUs information of delay and area. In a bitwidth-aware HLS, we have to obtain a multiple-bitwidth FUs information of delay and area. We can get it by the following ways:

1. Using modeling formulations to estimate FUs delay and area based on the bitwidth.
2. Performing characterization for all the kinds of FUs and all the numbers of bitwidths to generate FUs information beforehand.
3. Performing characterization dynamically, i.e, performing it when it becomes necessary.

Solution 1 is not a reasonable way since the relationship between bitwidths and delays/areas depends on the implementations of FUs and the target devices, thus there is no perfect modelings for estimating multiple-bitwidth FUs. Solution 2 is also unreasonable since combinations of all the kinds of FUs (e.g., adders and multipliers) and all the numbers of bitwidths are so large that it costs a long execution time if we perform the characterization for those beforehand. Therefore, we employ Solution 3 as an FU characterization policy in this chapter in order to save the HLS execution time.

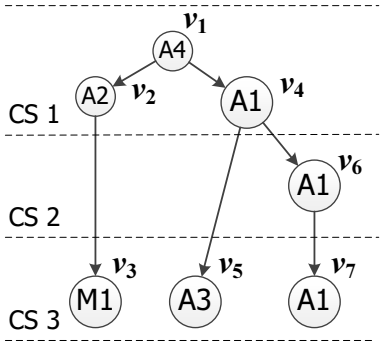


(a) DFG with bitwidths.

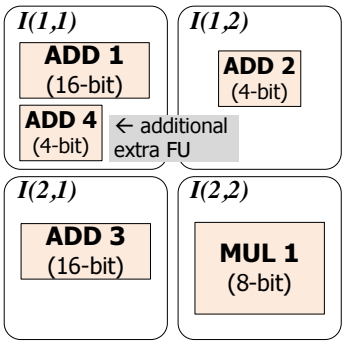
T_{clk}	3.0 ns
# of Islands	2×2
Area of an Island	240
# of FUs	Adder×3, Multiplier×1
Delay of a register	0.4 ns
Transfer time to adjacent islands (i.e., C_d)	1.0 ns

(b) The clock period and the specifications of RDR architecture.

Figure 4.4: Inputs to our HLS problem with multiple-operation chainings for RDR architecture.



(a) The scheduled and bound DFG.



(b) The FU floorplan result.

bitwidth	+	*
4	0.5 ns / 40	--
8	--	2.5 ns / 240
16	2.0 ns / 160	--

(c) The FU characterization (delay/area).

Figure 4.5: Outputs of our HLS problem with multiple-operation chainings for RDR architecture.

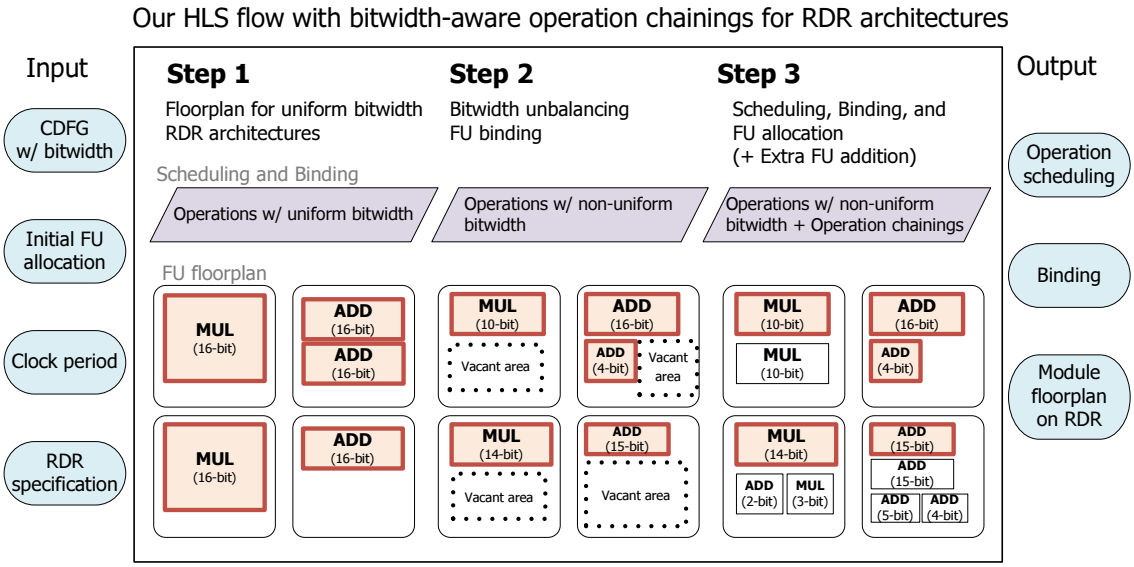


Figure 4.6: Overview of our synthesis flow.

4.4.2 Strategy

Let us consider several strategies to reduce the latency by applying operation chainings for RDR architectures.

In an HLS flow, we consider operation scheduling, binding, allocation, and floorplanning as in many HLS algorithms. In our target HLS problem, to realize operation chainings, operation scheduling and FU binding play the most important roles since we need to change them when constructing chainings. The conventional floorplan-aware HLS algorithm (Chapter 2) mainly focuses on operation scheduling and FU binding. However, we cannot take advantages of bitwidth-based FU cost (delay and area) only using the conventional approach (Chapter 2). In this chapter, in order to take care operation bitwidth, we optimize FU allocation and floorplan as well as operation scheduling and FU binding.

It is a complicated task to optimize floorplan-aware scheduling, FU binding, and their bitwidths simultaneously. We divide the synthesis flow into two tasks. Firstly, we obtain scheduling, binding, allocation, and floorplan result under uniform bitwidths (Step 1). Then, we optimize bitwidths of FUs followed by updating scheduling and binding (Step 2 and Step 3).

4.4.3 Synthesis Flow

Overview of our proposed HLS flow is shown in Fig. 4.6. Firstly, we obtain uniform bitwidth FU allocation and floorplan (Step 1). After that, the following two steps are

Table 4.1: SA parameters.

Parameter	Value
Initial temperature T	1000
Temperature updating	$T = 0.95 \cdot T$
Number of temperature annealing iterations (outer loop)	100 (at least)
Number of iterations at each temperature (inner loop)	1000
Weight parameter α in the cost function	0.9
Annealing exit condition	Continuous 100000 inner loops w/o decreasing the cost

performed: bitwidth unbalancing FU binding (Step 2) and multiple-bitwidth scheduling, binding, and extra FU allocation (Step 3). The key idea is to create vacant islands and place extra FUs into such islands to minimize the latency by realizing non-uniform-bitwidth operation chainings. Operation scheduling, binding, and floorplan (Step 3) performs followed by register binding and controller synthesis.

In this section, the details of key steps (Step 1 to 3) of our synthesis algorithm are described below.

Step 1: Initial FU Allocation and Floorplanning

In the initial step, we find the initial FU floorplan on RDR architectures. We obtain an allocation and a floorplan of FUs with uniform bitwidths using simulated annealing (SA) [32], which optimizes the latency and the wire length between islands by finding out the optimal allocation and floorplan. In this step, bitwidth of all operations are fixed to the maximum uniform bitwidth in the input application. This step generates an optimal scheduling and binding solution and an optimal floorplanning for uniform bitwidth operations.

Step 1 is based on "SA-based coarse placement engine" in "scheduling-driven placement" proposed in [15]. "Island" in our proposed algorithm corresponds to "bin structure" in [15]. In SA-based optimization in [15], they optimize placement of FUs onto islands to minimize the overall control steps. According to [15], let $C^c_{delay}(current)$ and $C^p_{delay}(previous)$ be current cost and previous cost of interconnection delay for all edges in DFG, respectively, and $C^c_{wire}(current)$ and $C^p_{wire}(previous)$ be current cost and previous cost of wire length, respectively. Cost function, $Cost$, of SA is represented by the weighted sum of the interconnection delay cost and the wire length cost as below:

$$Cost = \alpha \cdot \frac{C^c_{delay}}{C^p_{delay}} + (1 - \alpha) \cdot \frac{C^c_{wire}}{C^p_{wire}}, \tag{4.5}$$

where α is the weight. The detailed definitions of these variables are described in [15].

Step 2.1: Let $FU = \{fu_1, \ldots, fu_n\}$ be a set of functional units assigned to all the islands.

Step 2.2: For each control step cs and for each island t, perform (1)–(2) below:

(1): Sort the operation nodes in $V' \subseteq V$ scheduled at cs and bound to t, in the descending order of $b_{op}(v')$ value ($v' \in V'$).

(2): For each operation node $v' \in V'$, re-bind v' to FU $fu_i \in EFU(v')$ where fu_i is unused at cs and i is the smallest index in FU.

Step 2.3: For each functional unit $fu \in FU$, set FU bitwidth $b_{fu}(fu) = \max_{v'' \in V''}\{b_{op}(v'')\}$, where V'' is a set of operation nodes which is bound to fu in Step 2.2. If the FU fu is not used by any operations, fu is removed.

Figure 4.7: Algorithm of Step 2: Bitwidth unbalancing FU binding.

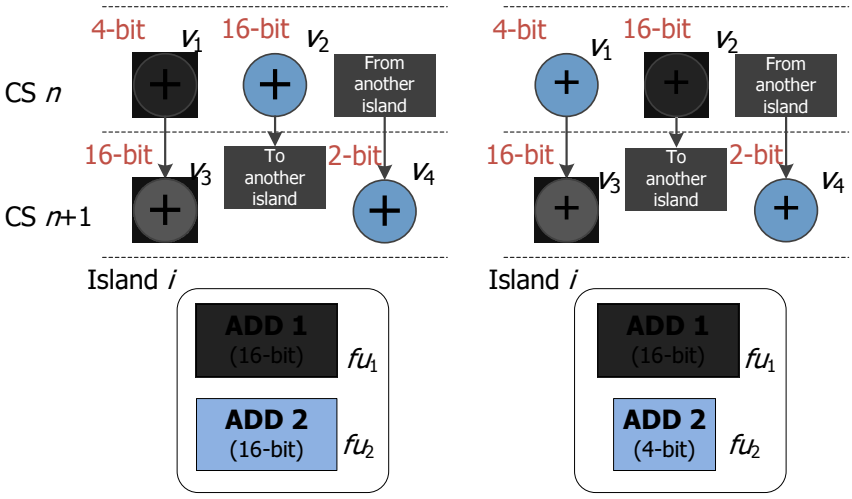
In our proposed algorithm, we also use the SA-based optimization algorithm [15]. Selection method of an adjacent solution and the cost function are the same as [15]. Table 4.1 shows the SA parameters we set as in [4, 57]. The annealing exits when a continuous 100000 inner loops without decreasing the cost is achieved (when the solution is sufficiently converged). As for the weight parameter α in the cost function, we set $\alpha = 0.9$ as in [4, 57] since we optimize total interconnection delays more than total wire lengths.

Step 2: Bitwidth Unbalancing FU Binding for RDR Architecture

Step 2 tries to adjust the bitwidth of FUs to eliminate the unused bits. The goal of this step is to resize them to the optimal bitwidth. FU binding plays an important and effective role to achieve the goal, because operations assignments to FUs strongly affect the required bitwidth of FUs. Assume that the maximum bitwidth for operations $v_i \in V$ executed on an FU fu is M bits, fu needs at most M bits then we can eliminate the unused bits if the FU fu has more than M bits initially.

In order to achieve the goal above, this step performs bitwidth unbalancing FU binding to reduce bitwidth of FUs efficiently. Inputs to Step 2 are DFG $G = (V, B, E)$, initial FU floorplan, initial scheduling result, and initial FU binding result. Outputs of Step 2 are FU floorplan in which FU bitwidths are reduced as much as possible and FU binding. The objective is to maximize the vacant area in every island. Figure 4.7 shows the algorithm of Step 2.

Figure 4.8 illustrates the algorithm of Step 2 shown in Fig. 4.7. Figure 4.8(a) shows a part of inputs to Step 2. In Step 2.1, we get $FU = \{fu_1, fu_2\}$. In Step 2.2, for island t, the following bindings are performed. When $cs = n$, operations in V' are sorted as



(a) Before performing Step 2: Initial binding is shown. FUs bitwidths cannot be reduced.

(b) After performing Step 2: FUs bitwidths are reduced.

Figure 4.8: Illustration of Step 2.

$V' = \{v_2, v_1\}$, then v_2 and v_1 are bound to ADD 1 (fu_1) and ADD 2 (fu_2), respectively. When $cs = n + 1$, operations in V' are sorted as $V' = \{v_3, v_4\}$, then v_3 and v_4 are bound to ADD 1 (fu_1) and ADD 2 (fu_2), respectively. In Step 2.3, for FU fu_1, FU bitwidth is calculated by $b_{fu}(fu_1) = 16$ and FU bitwidth is unchanged. For FU fu_2, FU bitwidth is calculated by $b_{fu}(fu_2) = 4$ and FU bitwidth is reduced to 4 bits. Note that if we bind v_1 and v_3 to ADD 1 (fu_1), and v_2 and v_4 to ADD 2 (fu_2), respectively, both FUs must have a bitwidth of 16, i.e., we obtain a worse result.

This algorithm ensures that, for any types of functional units, bitwidth of an FU with index i is greater than or equal to bitwidth of any FUs with indices i' ($i < i'$). The reason why we do not change operation schedulings in this algorithm is to prevent the overall latency from increasing caused by influences of the interconnection delays between islands.

Step 3: Multiple-Bitwidth Scheduling, Binding, and FU Allocation for RDR Architectures

Step 3 performs scheduling, binding, and FU allocation. Utilizing vacant islands generated by Step 2, we perform operation scheduling and binding while adding some extra FUs to realize bitwidth-aware operation chainings to optimize the latency of the input application.

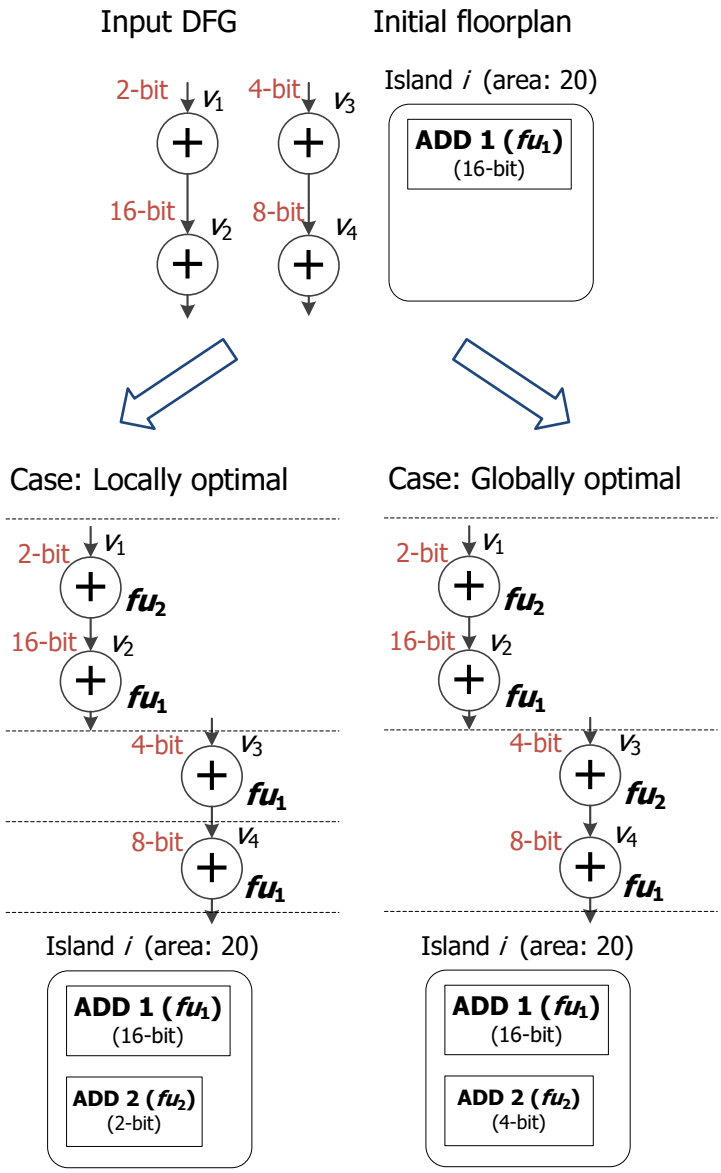


Figure 4.9: Strategy for adding extra FUs in Step 3.

Before scheduling and binding, we firstly perform profiling of the input application. We obtain general bitwidth information in the input application through this profiling. In this profiling, by traversing all operation nodes in the input application, frequency counts of each operation type and the bitwidth are recorded as "bitwidth frequency". For example, in Fig. 4.4(a), the "bitwidth frequency" for 4-bit '+' is 2 and this is represented as $bf('+', 4) = 2$. In the same way, we can express $bf('+', 16) = 4$, $bf('*', 8) = 1$, and $bf('*', 16) = 0$. These values are used in this step.

The scheduling is based on list-scheduling. Since we aim to schedule the operations

on the critical path including interconnection delays, the priorities of the operations in list-scheduling are set based on the critical path length with inter-islands delay, which is named cpl. Based on the algorithm in Chapter 2, the cpl from v to a primary output when a node v is bound to an FU fu_i is calculated by

$$cpl(v, fu_i) = d_{single}(fu_i) + \max_{v_k \in S(v)} \left\{ \min_{fu_j \in EFU(v_k)} \left\{ D_t(t_i, t_j) + cpl(v_k, fu_j) \right\} \right\} \tag{4.6}$$

where we assume fu_i and fu_j is placed on the island t_i and t_j, respectively, and $S(v)$ is a set of successor nodes of v. Then the priority of an operation v is calculated by

$$pr(v) = \min_{fu_i \in EFU(v)} \{cpl(v, fu_i)\} . \tag{4.7}$$

If the priority calculated by Eq. (4.7) is equal, an operation with larger bitwidth is prior to others. A ready list RL which contains ready operation nodes is implemented as a priority queue.

In our scheduling algorithm, we try to append some extra FUs into the vacant islands in the following way. Assume that an FU fu is placed in a island i. If $b_{op}(v) = b_{fu}(fu)$, v is scheduled/bound to cs/fu. If $b_{op}(v) < b_{fu}(fu)$, an FU fu' with $b_{fu}(fu') \in \left\{b_{op}(v), \ldots, M_b(op)\right\}$ bits can be added into the island i, where $M_b(op)$ is maximum bitwidth of the type op in the input DFG. If there exists enough area, i.e., $a_{fu'} \leq A - A_i$, we can append a new FU fu' as an additional extra FU. Bitwidth of fu' is determined based on "bitwidth frequency" described above. Reason why we do not append an FU whose bitwidth is exactly the same as $b_{op}(v)$ is that an FU with $b_{op}(v)$ bitwidth is considered locally optimal here, but an FU with more bitwidth can have more opportunities to be bound by other operation nodes and this can be globally optimal. Since the characterization time for a new FU whose cost (delay and area) is unknown costs much as described in Section 4.4.1, we want to keep the number of such characterizations small as much as possible. Therefore, we determine bitwidth of the additional extra FU in the following calculation: Let $needed(op, b)$ be a needed count of FUs for executing operation op with b-bit. $needed(op, b)$ is calculated by

$$needed(op, b) = bf(op, b) - bound(op, b) - \sum_{b \leq b'} added(op, b') \tag{4.8}$$

where, $bf(op, b)$ is "bitwidth frequency" of b-bit operation op defined above, $bound(op, b)$ is the number of b-bit operations op which have been already scheduled and bound, and $added(op, b)$ is the number of b-bit FUs added which can execute the operation op. The bitwidth b' which takes the maximum counts in $needed(op, b')$, i.e., b' in

$$\arg\max_{1 \leq b' \leq M_b(op)} \{needed(op, b')\} \tag{4.9}$$

is determined as the bitwidth of the additional FU.

When we can add a required extra FU onto an island, we do not always determine the FU bitwidth which is exactly the same as the FU bitwidth just required. Our strategy adds an extra FU whose bitwidth is calculated by Eq. (4.8) and Eq. (4.9) based on our application profiling. The calculated bitwidth may be greater than the FU bitwidth just required. Let us explain that this strategy can avoid a locally optimal solution.

An example is show in Fig. 4.9. To explain the example, for simplicity, assume that the clock period is 2.0ns, area of an island is 20, and delay/area of 16-bit, 4-bit, and 2-bit adders are 1.6ns/16, 0.4ns/4, and 0.2ns/2, respectively. An FU delay assumes to include register delay and MUX delay. Now, DFG and initial FUs floorplan are given as shown in the figure. The priority of v_1 assumes to be higher than priority of v_3 in scheduling. In list-scheduling, after picking up an addition v_1, an extra FU addition onto the island t is tried in Step 3.5-(2). If we add an extra FU fu_2 with $b_{op}(v_1) = 2$ bits which is same as an addition v_1 (locally optimal), either v_3 or v_4 cannot be bound to fu_2 but both of them will be bound to fu_1 instead. In this case, a scheduling result with 3 control steps will be obtained. On the other hand, if we add an extra FU fu_2 whose bitwidth is set to be 4 bits according to Eq. (4.8) and Eq. (4.9) (globally optimal), v_3 and v_4 are bound to fu_2 and fu_1, respectively, and a scheduling result with 2 control steps will be obtained.

Figure 4.10 shows the algorithm of Step 3. Inputs to Step 3 are DFG $G = (V, B, E)$ and FU floorplan which contains some vacant areas obtained by Step 2. Outputs of Step 3 are Operations scheduling, FU floorplan, and FU binding. The objective is to minimize the latency of the input application. Step 3 is based on the list-scheduling to minimize the critical path which includes the interconnection delays under the FUs constraint.

In our scheduling algorithm, we initially set $cs = 1$. For each control step cs, we pick up an operation v with the largest priority from RL. If operation is ready to be scheduled, i.e., inputs data of v are ready, v is tried to scheduled/bound to cs/fu having the smallest possible $cpl(v, fu)$.

In the same cs, RL is re-constructed iteratively, and we continue scheduling until no operations are scheduled. When the parents operations of v are already scheduled in the same cs, a cluster including v and its parents is realized as multiple-bitwidth operation chainings if it satisfy Eq. (4.4). Unlike the conventional algorithm (Chapter 2) which construct operation chainings with successors greedily, our algorithm realizes operation chainings based on cpl so chainings are realized on only the critical path(s). Because of that, when there is no additional extra FUs, more optimized operation chainings solution can be obtained compared to Chapter 2.

The idea of introducing cpl and node priority is similar to that of Chapter 2, but we newly introduce in our algorithm the idea of adding extra FUs and constructing operation chainings in Step 3. The main differences between our proposed algorithm and the algorithm in Chapter 2 are:

Step 3.1: Perform application profiling and obtain the "bitwidth frequency" as described in Section 4.4.3.

Step 3.2: Construct a ready-list RL by inserting initial ready nodes into RL.

Step 3.3: If RL is empty, finish the scheduling.

Step 3.4: For all unscheduled nodes, (re-)calculate priority by Eq. (4.7).

Step 3.5: For every node v in RL in the descending order of its priority, perform (1)–(5) below:

(1): Choose an FU fu having the smallest possible $cpl(v, fu)$ among currently available FUs, and let t be the island on which fu is placed. If there is no such FU, skip (2)–(5) below.

(2): If $b_{op}(v) < b_{fu}(fu)$, try adding an extra FU for v onto t as described in Section 4.4.3. If it succeeds, let fu be the added FU and obtain delay and area of functional unit fu using logic synthesizer (described in Section 4.4.1).

(3): If at least one of the parent(s) of v are already scheduled to cs, and the cluster composed of such parent(s) and v satisfies Eq. (4.4), v is scheduled/bound to cs/fu (multiple-bitwidth operation chaining is succeeded).

(4): Otherwise, v is tried to scheduled/bound to cs/fu, and if data from all parent(s) of v is transferable to v, then v is scheduled/bound to cs/fu.

(5): If the scheduling and binding of v is succeeded, remove v from RL. Insert new ready node(s) into RL.

Step 3.6: $cs = cs + 1$. Go to Step 3.3.

Figure 4.10: Algorithm of Step 3: Scheduling, binding, and FU allocation for RDR architectures.

(1) Our proposed algorithm appends some extra FUs with appropriate bitwidths into the vacant islands which are obtained in Step 2 (Step 3.1 and Step 3.5-(2)).

(2) The algorithm in Chapter 2 only realizes operation chainings on consecutive two operations. However, our proposed algorithm can realize operation chainings on clusters which include the operations in the critical path based on Eq. (4.4) (Step 3.5-(3)).

Finally after this step, we obtain an optimized bitwidth-aware scheduling and FU binding solution with operation chainings and an FU floorplan solution for RDR architectures.

Table 4.2: Benchmark applications.

Application name	#operations
Auto Regression Filter (ARF)	28
Discrete Cosine Transform (DCT)	48
EPIC	56
Elliptic Wave Filter (EWF)	34
Elliptic Wave Filter (EWF3)	102
Finite Impulse Response Filter (FIR)	75
JPEG—BMP Header (JPEG-BMPH)	106
JPEG—Forward DCT (JPEG-FDCT)	134
JPEG—Inverse DCT (JPEG-IDCT)	122
JPEG—Smooth Downsample (JPEG-SD)	51
MESA—Horner Bezier (MESA-HB)	18
MESA—Interpolate Aux (MESA-IA)	108
MESA—Matrix Multiplication (MESA-MM)	109
MESA—Smooth Triangle (MESA-ST)	197

Table 4.3: FUs area.

FU	Normalized area (16-bit)
Adder	0.44
Subtractor	0.45
Multiplier	1.00
Right shifter	0.46
Left shifter	0.44
Comparator	0.39
And	0.37

4.5 Experimental Results

In this section, the experimental results are shown, and some discussions are described.

4.5.1 Experimental Setup

We have implemented our proposed algorithm in C++ on Intel Xeon CPU E7-4870 2.4 GHz $\times$ 40 machine with 330GB memory. We applied our proposed algorithm to 14 benchmark applications. The benchmark applications are summarized in Table 4.2. These benchmarks are derived from ExPRESS benchmarks [25]. Each operation have different bitwidth. For our proposed algorithm, we give bitwidths of 2, 4, 6, or 8-bit

Table 4.4: Delay and area of registers.

Bitwidth	Delay [ns]	Area [μm^2]
1	0.11	13
2	0.11	26
3	0.11	40
4	0.11	53
5	0.11	67
6	0.11	80
7	0.11	93
8	0.11	107
9	0.11	120
10	0.11	134
11	0.11	147
12	0.11	160
13	0.11	174
14	0.11	187
15	0.11	201
16	0.11	214

to PI operations, and set bitwidths for other operations in the following manner: output bitwidth of multiplier is sum of the two inputs' bitwidth, output bitwidth of the other FUs (e.g., adder and subtractor) is maximum of the two inputs' bitwidth. We assume uniform 16-bit operations for the evaluation of the conventional approach. Synopsys Design Compiler under 90nm technology node is used as the logic synthesis tool to evaluate delay and area of FUs as well as controller (FSM) synthesis. The interconnection delay is assumes to be in proportion to the distance and takes 1ns for 250μm according to [45]. In case an application needs memory accesses, a memory port is placed in one of the islands, and the number of memory port is one.

We set the other parameters in the following manners.

Clock Period

According to a previous research (Chapter 2), which used the same manufacturing process rule (90nm technology node), sum of FU's delay (which includes MUXs' delay) and register's delay is 2.93ns at most. Thus, we set the basic clock period to be 3.0ns. For larger applications, however, we set the clock period to be 4.5ns as described just below.

Table 4.5: Delay and area of 2-to-1 MUXs.

Bitwidth	Delay [ns]	Area [μm^2]
1	0.04	7
2	0.04	14
3	0.04	21
4	0.04	28
5	0.04	35
6	0.04	42
7	0.04	49
8	0.04	56
9	0.04	63
10	0.04	70
11	0.04	77
12	0.04	84
13	0.04	91
14	0.04	98
15	0.04	105
16	0.04	112

Size of an Island

According to Chapter 2, functional unit with the largest delay is the multiplier and the subtractor is the second. We assume that the number of multipliers in an island is one or zero as in Chapter 2, and it is always placed adjacent to registers, so that we can ignore the interconnection delays for multipliers. Let us consider the worst case of interconnection delays for the other FUs. According to Chapter 2, delay of subtractor is 1.45ns and that of register is 0.11ns, thus their total logic delay is 1.56ns. When the clock period is 3.0ns, allowable time for interconnection is $3.0 - 1.56 = 1.44$ ns. Now, since we assume that the interconnection delay is 1ns for each 250μm wire length, we can transfer data to $250 \times 1.44 = 360\mu$m away for 1.44ns. Therefore, when the clock period is 3.0ns, we set the square island size to be $360/4 = 90\mu$m.

However, for the seven applications which have more than 100 operations in Table 4.2, we set the clock period to be 4.5ns. When the clock period is 4.5ns, by a similar discussion above, allowable time for interconnection is $4.5 - 1.56 = 2.94$ ns. We can transfer data to $250 \times 2.94 = 735\mu$m away for 2.94ns. Therefore, we need to set the island size not to exceed $735/4 = 183.75\mu$m. Then we set it to be 150μm.

Number of Islands

Similar to the conventional research (Chapter 2), we set the number of islands to be 2×2 or 2×3. To show that our proposed algorithm is efficient not only a particular "number of islands," we added results for 2×2 and 2×3 for all the benchmark applications. Note that, we added only 2×3 for EPIC and JPEG-BMPH as described just below.

Number of Initial Functional Units

Table 4.3 shows 16-bit FUs areas. The areas are normalized to the area of a multiplier which has the largest area among all the FUs. We assume that the number of multipliers in an island is one or zero as in Chapter 2.

As we can see the experimental results in Chapter 2, it is appropriate to determine the "number of available operational units" by the following manner.

The main idea is that the initial numbers of FUs with each type should be set in proportion to the corresponding operations with the same type in the input application. Assume that every initial FU with type op has an identical area. For example, we consider a 16-bit adder for an addition initially. Then we firstly calculate the ratio P_{op} of the operations with each type op to all the operations in every application. Let OP be a set of all operation types, A_{op} be a normalized area of FU with type op, T be a total number of islands, we find the maximum scale factor n_{max} so that the total area does not exceed the total islands area, which satisfies

$$\sum_{op \in OP} \lceil \lceil P_{op} \cdot n_{max} \rceil \cdot A_{op} \rceil \leq T. \tag{4.10}$$

A_{op} is given by Table 4.3. Then the initial number of FUs with each type op is set to be $\lceil P_{op} \cdot n_{max} \rceil$.

For example, we set the initial numbers of FUs for EWF and islands 2×2 (total islands $T = 4$) in the following way. EWF consists of 34 operations. 26 of those are additions and 8 of those are multiplications. Thus, the ratios of operation types are calculated as $P_{add} = 26/34 = 0.76$ and $P_{mul} = 8/34 = 0.24$, respectively. The maximum scale factor n_{max} which satisfies Eq. (4.10) is $n_{max} = 5$ as in

$$\sum_{op \in OP} \lceil \lceil P_{op} \cdot n \rceil \cdot A_{op} \rceil = \lceil \lceil 0.76 \cdot 5 \rceil \cdot 0.44 \rceil + \lceil \lceil 0.24 \cdot 5 \rceil \cdot 1.0 \rceil = 4. \tag{4.11}$$

Therefore, we have:

- Number of adders: $\lceil 0.76 \cdot 5 \rceil = 4$.
- Number of multipliers: $\lceil 0.24 \cdot 5 \rceil = 2$.

Then we give (A×4, M×2) to EWF for 2×2 islands in Table 4. Note that, by employing this way, the obtained initial numbers of FUs with each type, which are four adders and two multipliers, are almost the same as the ratios of the operations with each type in EWF, which are 0.76 (additions) and 0.24 (multiplications).

For the other applications except for EPIC and JPEG-BMPH, initial numbers of FUs are determined in the same way for islands 2×2 and 2×3. For EPIC and JPEG-BMPH, there is no scale factor n which satisfies Eq. (4.10) for the islands 2×2, thus we show the results only for the islands 2×3.

The results would be improved if we optimize these parameters, which is one of the future works.

Delay and Area of Modules

For functional units and controllers, we obtain delay and area using a logic synthesis tool (Synopsys Design Compiler 2012.06 under 90nm technology node) through our HLS synthesis flow. For functional units, the logic synthesizer tries to minimize the delay to minimize the number of control steps obtained. For controllers, the logic synthesizer tries to minimize the area under the given clock period timing constraint.

For registers and MUXs, we prepare the delay and area for each bitwidth as beforehand. This information is used in our HLS flow. Table 4.4 and Table 4.5 show delay and area information of registers and MUXs, respectively.

4.5.2 Results and Discussion

To show the effectiveness of our proposed algorithm, we compared two algorithms below:

HLS with operation chainings for uniform bitwidth (Chapter 2): The conventional HLS algorithm for RDR architecture with operation chainings proposed in Chapter 2. Note that we modified the delay estimation to the model where the interconnection delay is in proportion to the distance.[4]

Proposed: Our proposed algorithm with bitwidth-aware operation chainings described in Section 4.4.

The experimental results are shown in Table 4.6 and Table 4.7. "Latency" column shows the overall latency and we find out that our proposed algorithm reduces the latency compared to the conventional algorithm in Chapter 2 for all experiments except for FIR (2×3). Note that for FIR (2×3), our proposed algorithm does not increase the latency compared to Chapter 2. Our algorithm reduces the latency by up to 47% compared

[4]More details are described in Section 4.5.5.

to the algorithm in Chapter 2 for EWF, by 17% on average. "Additional extra FUs" column shows the number of extra FUs added into the vacant islands in Step 3. In this results, we can confirm the proposed algorithm can reduce the latency by good work of the redundant bitwidth elimination (Step 2) and extra FUs adding (Step 3). We can see in some cases the addition of extra FUs helps the construction of operation chainings, which reduces the latency. Note that our scheduling and binding algorithm can reduce the latency by realizing operation chainings even if there is no extra FUs are added.

"#registers" and "#MUX" in Table 4.6 and Table 4.7 are the number of generated registers and 2-to-1 MUXs without taking into consideration of bitwidths. This means, for example, if our proposed algorithm generates a 2-bit register and a 16-bit register, the number of registers becomes two. To show more detailed results, "#registers (1-bit)" and "#MUX (1-bit)" columns in Table 4.6 and Table 4.7 represent the number of registers and 2-to-1 MUXs counted by their bitwidths. For example, if our proposed algorithm generates a 2-bit register and a 16-bit register, the values are counted by 2 and 16, respectively.

Number of registers ("#registers") is reduced by up to 32% and 4% on average. Number of MUXs ("#MUX") is reduced by up to 54% and 13% on average.

As seen in the result, even if our proposed algorithm increases the number of registers and/or MUXs, our proposed algorithm does not increase the number of total bitwidths in any cases. Therefore, we conclude that the number of registers and the number of MUXs obtained by our algorithm are the same or reduced compared to the algorithm in Chapter 2 from the bitwidth point of view in all the cases, which means the area of registers and MUXs can be reduced.

However, in some cases, our proposed algorithm increases the number of registers and/or MUXs as shown in "#registers" and "#MUX" (without taking into consideration of bitwidths). The qualitative reasons are discussed below:

Reason why the number of registers may be increased:

Since a registers binding result is obtained by our proposed algorithm which is different from a result by the algorithm in Chapter 2, the number of registers may change. In particular, when the operations, which are calculated within the same island by the algorithm in Chapter 2, are calculated in different islands by our proposed algorithm, the total number of registers may be increased compared to the algorithm in Chapter 2.

Reason why the number of MUXs may be increased:

Similarly, FUs/registers binding results are obtained by our proposed algorithm which are different from results by Chapter 2. Since the FUs/registers sharings are different between Chapter 2 and our proposed algorithm, the total number of MUXs may be increased compared to Chapter 2.

Table 4.6: Experimental results (1/2).

App.	#islands	Initial FUs[****]	Algorithm	Additional extra FUs	Clock period [ns]	Control steps	Latency [ns]	#registers	#registers (1-bit)	#MUXs	#MUXs (1-bit)
MESA-HB	2×2	A ×2, M ×3, Mem ×1	Ch. 2	–	3.0	12	36.0 (1.00)	7 (1.00)	112 (1.00)	13 (1.00)	208 (1.00)
			Proposed	A×1, M×2	3.0	8	24.0 (0.67)	7 (1.00)	50 (0.45)	6 (0.46)	34 (0.16)
MESA-HB	2×3	A ×4, M ×4, Mem ×1	Ch. 2	–	3.0	12	36.0 (1.00)	9 (1.00)	144 (1.00)	11 (1.00)	176 (1.00)
			Proposed	–	3.0	10	30.0 (0.83)	7 (0.78)	112 (0.78)	9 (0.82)	144 (0.82)
ARF	2×2	A ×2, M ×3	Ch. 2	–	3.0	12	36.0 (1.00)	11 (1.00)	176 (1.00)	26 (1.00)	416 (1.00)
			Proposed	–	3.0	10	30.0 (0.83)	11 (1.00)	176 (1.00)	24 (0.92)	384 (0.92)
ARF	2×3	A ×3, M ×4	Ch. 2	–	3.0	11	33.0 (1.00)	12 (1.00)	192 (1.00)	26 (1.00)	416 (1.00)
			Proposed	–	3.0	8	24.0 (0.73)	12 (1.00)	192 (1.00)	19 (0.73)	304 (0.73)
EWF	2×2	A ×4, M ×2	Ch. 2	–	3.0	15	45.0 (1.00)	12 (1.00)	192 (1.00)	36 (1.00)	576 (1.00)
			Proposed	A×2	3.0	8	24.0 (0.53)	10 (0.83)	74 (0.39)	22 (0.61)	166 (0.29)
EWF	2×3	A ×7, M ×2	Ch. 2	–	3.0	15	45.0 (1.00)	12 (1.00)	192 (1.00)	36 (1.00)	576 (1.00)
			Proposed	A×2	3.0	9	27.0 (0.60)	11 (0.92)	112 (0.58)	19 (0.53)	152 (0.26)
DCT	2×2	A ×4, M ×2	Ch. 2	–	3.0	11	33.0 (1.00)	21 (1.00)	336 (1.00)	64 (1.00)	1024 (1.00)
			Proposed	A×1, M×3	3.0	7	21.0 (0.64)	22 (1.05)	260 (0.77)	64 (1.00)	762 (0.74)
DCT	2×3	A ×6, M ×3	Ch. 2	–	3.0	9	27.0 (1.00)	27 (1.00)	432 (1.00)	59 (1.00)	944 (1.00)
			Proposed	A×1, M×1	3.0	7	21.0 (0.78)	22 (0.81)	308 (0.71)	60 (1.02)	852 (0.90)
JPEG-SD	2×2	A ×4, M ×1, R ×1, Mem ×1	Ch. 2	–	3.0	25	75.0 (1.00)	19 (1.00)	304 (1.00)	39 (1.00)	624 (1.00)
			Proposed	A×4, M×2	3.0	20	60.0 (0.80)	19 (1.00)	182 (0.60)	36 (0.92)	366 (0.59)
JPEG-SD	2×3	A ×9, M ×1, R ×1, Mem ×1	Ch. 2	–	3.0	25	75.0 (1.00)	18 (1.00)	288 (1.00)	38 (1.00)	608 (1.00)
			Proposed	A×2	3.0	23	69.0 (0.92)	17 (0.94)	234 (0.81)	40 (1.05)	528 (0.87)
EPIC	2×3	A ×2, S ×1, M ×1, R ×1, L ×1, Mem ×1	Ch. 2	–	3.0	21	63.0 (1.00)	14 (1.00)	224 (1.00)	32 (1.00)	512 (1.00)
			Proposed	A×3, M×3	3.0	20	60.0 (0.95)	14 (1.00)	92 (0.41)	32 (1.00)	208 (0.41)
FIR	2×2	A ×2, M ×3, Mem ×1	Ch. 2	–	3.0	22	66.0 (1.00)	21 (1.00)	336 (1.00)	51 (1.00)	816 (1.00)
			Proposed	A×2	3.0	20	60.0 (0.91)	20 (0.95)	244 (0.73)	65 (1.27)	804 (0.99)
FIR	2×3	A ×4, M ×4, Mem ×1	Ch. 2	–	3.0	16	48.0 (1.00)	19 (1.00)	304 (1.00)	46 (1.00)	736 (1.00)
			Proposed	–	3.0	16	48.0 (1.00)	13 (0.68)	200 (0.66)	41 (0.89)	592 (0.80)

[****] FUs are shown in the shorten names: A: Adder, S: Subtractor, M: Multiplier, R: Right shifter, L: Left shifter, C: Comparator, And: And unit, Mem: Memory port.

Table 4.7: Experimental results (2/2).

App.	#islands	Initial FUs****	Algorithm	Additional extra FUs	Clock period [ns]	Control steps	Latency [ns]	#registers	#registers (1-bit)	#MUXs	#MUXs (1-bit)
EWF3	2 × 2	A ×4, M ×2	Ch. 2	–	4.5	31	139.5 (1.00)	14 (1.00)	224 (1.00)	88 (1.00)	1408 (1.00)
			Proposed	–	4.5	24	108.0 (0.77)	14 (1.00)	224 (1.00)	78 (0.89)	1248 (0.89)
EWF3	2 × 3	A ×7, M ×2	Ch. 2	–	4.5	30	135.0 (1.00)	17 (1.00)	272 (1.00)	89 (1.00)	1424 (1.00)
			Proposed	–	4.5	23	103.5 (0.77)	15 (0.88)	240 (0.88)	78 (0.88)	1248 (0.88)
JPEG-BMPH	2 × 3	A ×4, M ×1, R ×2, C ×1, And ×2, Mem ×1	Ch. 2	–	4.5	36	162.0 (1.00)	17 (1.00)	272 (1.00)	26 (1.00)	416 (1.00)
			Proposed	A×3	4.5	35	157.5 (0.97)	16 (0.94)	192 (0.71)	17 (0.65)	228 (0.55)
MESA-IA	2 × 2	A ×1, S ×1, M ×2, Mem ×1	Ch. 2	–	4.5	28	126.0 (1.00)	27 (1.00)	432 (1.00)	70 (1.00)	1120 (1.00)
			Proposed	A×2, M×3	4.5	20	90.0 (0.71)	33 (1.22)	350 (0.81)	95 (1.36)	1032 (0.92)
MESA-IA	2 × 3	A ×4, S ×1, M ×3, Mem ×1	Ch. 2	–	4.5	21	94.5 (1.00)	25 (1.00)	400 (1.00)	94 (1.00)	1504 (1.00)
			Proposed	A×2, M×2	4.5	18	81.0 (0.86)	27 (1.08)	389 (0.97)	102 (1.09)	1494 (0.99)
MESA-MM	2 × 2	A ×3, M ×2, Mem ×1	Ch. 2	–	4.5	26	117.0 (1.00)	30 (1.00)	480 (1.00)	98 (1.00)	1568 (1.00)
			Proposed	A×1	4.5	25	112.5 (0.96)	27 (0.90)	392 (0.82)	67 (0.68)	1016 (0.65)
MESA-MM	2 × 3	A ×5, M ×3, Mem ×1	Ch. 2	–	4.5	26	117.0 (1.00)	28 (1.00)	448 (1.00)	91 (1.00)	1456 (1.00)
			Proposed	–	4.5	25	112.5 (0.96)	27 (0.96)	432 (0.96)	63 (0.69)	1008 (0.69)
JPEG-IDCT	2 × 2	A ×1, S ×1, M ×1, R ×1, Mem ×1	Ch. 2	–	4.5	46	207.0 (1.00)	28 (1.00)	448 (1.00)	116 (1.00)	1856 (1.00)
			Proposed	–	4.5	44	198.0 (0.96)	28 (1.00)	448 (1.00)	96 (0.83)	1536 (0.83)
JPEG-IDCT	2 × 3	A ×2, S ×2, M ×2, R ×2, Mem ×1	Ch. 2	–	4.5	30	135.0 (1.00)	33 (1.00)	528 (1.00)	136 (1.00)	2176 (1.00)
			Proposed	A×1	4.5	27	121.5 (0.90)	31 (0.94)	496 (0.94)	97 (0.71)	1552 (0.71)
JPEG-FDCT	2 × 2	A ×2, S ×1, M ×1, R ×1, Mem ×1	Ch. 2	–	4.5	40	180.0 (1.00)	23 (1.00)	368 (1.00)	94 (1.00)	1504 (1.00)
			Proposed	A×5, M×2	4.5	26	117.0 (0.65)	23 (1.00)	220 (0.60)	91 (0.97)	836 (0.56)
JPEG-FDCT	2 × 3	A ×4, S ×1, M ×2, R ×1, Mem ×1	Ch. 2	–	4.5	29	130.5 (1.00)	33 (1.00)	528 (1.00)	136 (1.00)	2176 (1.00)
			Proposed	A×1	4.5	26	117.0 (0.90)	26 (0.79)	400 (0.76)	105 (0.77)	1640 (0.75)
MESA-ST	2 × 2	A ×2, S ×1, M ×2, Mem ×1	Ch. 2	–	4.5	52	234.0 (1.00)	42 (1.00)	672 (1.00)	150 (1.00)	2400 (1.00)
			Proposed	–	4.5	51	229.5 (0.98)	43 (1.02)	656 (0.98)	136 (0.91)	2152 (0.90)
MESA-ST	2 × 3	A ×3, S ×1, M ×3, Mem ×1	Ch. 2	–	4.5	52	234.0 (1.00)	51 (1.00)	816 (1.00)	137 (1.00)	2192 (1.00)
			Proposed	A×3	4.5	51	229.5 (0.98)	62 (1.22)	742 (0.91)	146 (1.07)	1708 (0.78)

**** FUs are shown in the shorten names: A: Adder, S: Subtractor, M: Multiplier, R: Right shifter, L: Left shifter, C: Comparator, And: And unit, Mem: Memory port.

Table 4.8: Comparison between the delay models.

App.	#islands	Initial FUs	Delay model	Additional extra FUs	Clock period [ns]	Control steps	Latency [ns]	#registers	#registers (1-bit)	#MUXs	#MUXs (1-bit)
ARF	2×2	A ×2, M ×3	Linear	–	3.0	10	30.0	11	176	24	384
			Elmore	–	3.0	9	27.0	12	188	25	400
ARF	2×3	A ×3, M ×4	Linear	–	3.0	8	24.0	12	192	19	304
			Elmore	–	3.0	8	24.0	12	192	22	352
MESA-ST	2×2	A ×2, S ×1, M ×2, Mem ×1	Linear	–	4.5	51	229.5	43	656	136	2152
			Elmore	–	4.5	51	229.5	43	656	136	2152
MESA-ST	2×3	A ×3, S ×1, M ×3, Mem ×1	Linear	A×3	4.5	51	229.5	62	742	146	1708
			Elmore	A×3	4.5	51	229.5	62	742	146	1708

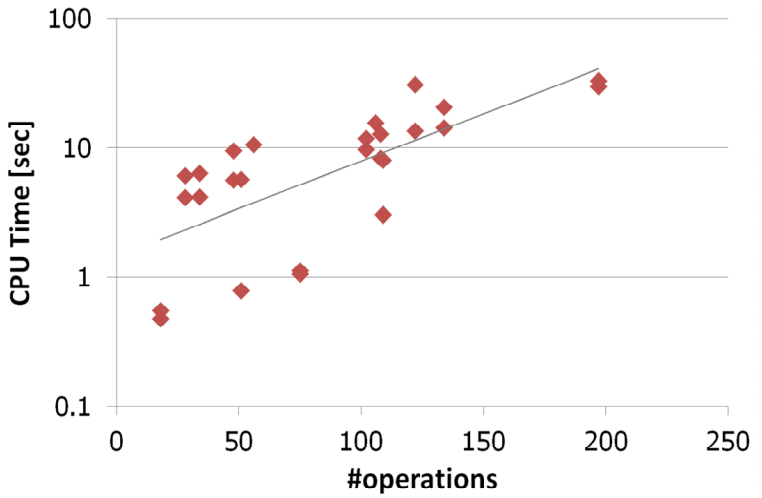


Figure 4.11: CPU time of Step 1.

4.5.3 Computation Time

As described in Section 4.4, our proposed algorithm consists of three steps, Step 1 to Step 3. Step 1 is based on [15], and SA is used only in this step. As described in Section 4.4.3, the annealing exits when a continuous 100000 inner loops without decreasing the cost is achieved (when the solution is sufficiently converged). The CPU time of Step 1 indicates the whole computation time of the annealing. For MESA-ST (197 operations) which has the largest number of operations among benchmark applications that we used, CPU time of Step 1 is 33.06sec, which takes 98.31% CPU time of all the steps. It is obvious that Step 1 is a dominant step in terms of CPU time.

Figure 4.11 shows the scalability of Step 1. In Fig. 4.11, the relations between the number of operations and the CPU time of Step 1 for all benchmark applications are plotted. The horizontal axis represents the number of operations in applications, and the vertical axis represents the CPU time of Step 1 in logarithmic scale. We can see that the CPU time of Step 1 tends to increase exponentially against the number of operations. We can conclude that the scalability of Step 1 is not good.

When we apply our proposed algorithm to larger applications in the future, Step 1 can be a bottleneck in terms of the execution time. In these situations, Step 1 cannot optimize the FUs floorplan sufficiently, and could affect bad influences to Step 2 and/or Step 3. We then need to improve the SA-based algorithm based on [15] in the future.

4.5.4 Possible Improvement of the Floorplanning

We can easily consider an improvement of our proposed HLS algorithm for a different DR architecture such as HDR architectures [2]. Similar to RDR architectures, HDR

Table 4.9: SA parameters for "Rectangle Packing Problem."

Parameter	Value
Initial temperature T	1000
Temperature updating	$T = 0.999 \cdot T$
Number of temperature annealing iterations	1000 (at least)
Annealing exit condition	Continuous 1000 inner loops w/o decreasing the area (i.e., solution is sufficiently converged)

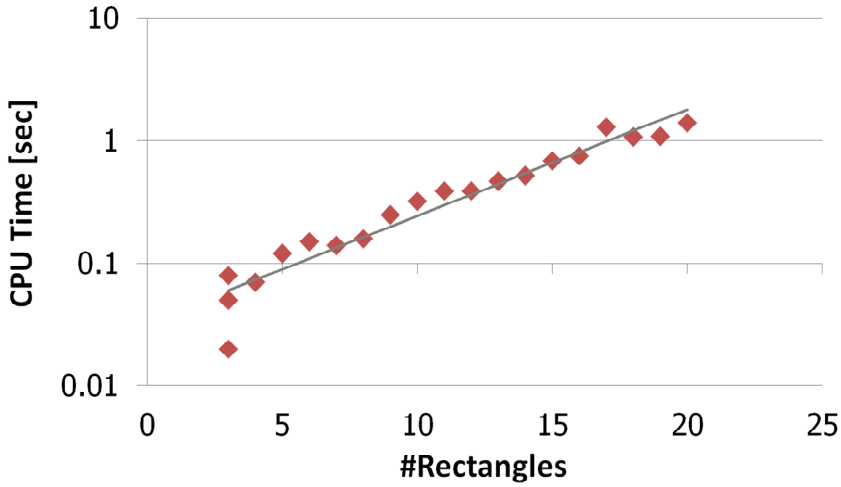


Figure 4.12: CPU time of simulated annealing for "Rectangle Packing Problem" if we consider extending our proposed HLS algorithm for HDR architectures.

architecture [2] is also one of DR architectures but does not always have islands with the same size, while RDR architecture has islands with the same size. We can use HDR architectures to reduce the vacant area of islands in RDR architectures, since we can fit the size of each island depending on the use of the island in HDR architectures. In an HLS algorithm targeting HDR architectures, placing FUs without overlaps to minimize the boundary box area (chip area) is defined as the floorplanning problem. In this problem, an FU can be modeled as a rectangle, and we can formulate the problem as "Rectangle Packing Problem." "Rectangle Packing Problem" can be also optimized with simulated annealing (SA) by using the floorplan representation called sequence-pair [44].

We have also implemented this SA-based algorithm by using sequence-pair for "Rectangle Packing Problem." We have implemented it in C++ on CentOS 6.8 and Intel Xeon CPU E5-2680 v3 2.50GHz × 40 machine with 270GB memory. Table 4.9 shows the SA parameters for "Rectangle Packing Problem." CPU time of this SA in the range of $3 \leq N \leq 20$ (where N is the number of rectangles) is measured and the results are shown in Fig. 4.12. As seen in Fig. 4.12, the CPU time of the SA also tends to increase

exponentially against the number of rectangles as same as in Fig. 4.11. We can conclude that the scalability of the floorplanning is not good if we consider extending our proposed HLS algorithm in this chapter for another DR architecture such as HDR architecture.

4.5.5 Comparisons to the Conventional Algorithms

We compare our proposed algorithm to the several conventional algorithms.

Comparison Between Delay Models

The algorithm in Chapter 2 assumes that the interconnection delay is in proportion to the square of the distance based on Elmore delay model. However, our proposed algorithm assumes that the interconnection delay is in proportion to the distance. The reason why we do so is that, in real designs, we can treat that the delay is in proportion to the distance by inserting buffers onto the wires properly as described in [14]. Buffer area is small enough and ignore it based on [14]. To clarify the influence caused by differences between these delay models, we picked up some applications from benchmark applications from Table 4.2 and compared two delay models below:

- The interconnection delay is in proportion to the distance (Linear),
- The interconnection delay is in proportion to the square of the distance (Elmore).

We picked up ARF which has the smallest number of operations among we used (28 operations) and MESA-ST which has the largest number of operations (197 operations). For these applications, the comparison results between the two delay models are shown in Table 4.8. As seen in the result, control steps are the same except for ARF (2×2), and there are almost no differences in the number of registers and the number of MUXs. Therefore, we can conclude that the difference of using these two delay estimation models will not affect the results.

Note that "Chapter 2" in the experimental results in Table 4.6 and Table 4.7 uses the modified interconnection delay estimation model where the delay assumes to be in proportion to the distance, while the original algorithm in Chapter 2 uses the model where the delay assumes to be in proportion to the square of the distance. Therefore, the conditions of interconnection delay estimation is equivalent in this chapter and the comparison is considered to be fair.

Comparison to [17]

We discuss the qualitative comparison between [17] and our proposed algorithm. [17] proposes a bitwidth-aware HLS algorithm targeting DR architectures which are similar

to our targeting RDR architectures. Their algorithm considers that the FUs with different bitwidths have different areas but have the same delays. It only focuses on the area minimization and does not realize operation chainings since the operation delays are assumed to be the same in every bitwidth. Therefore, our proposed algorithm has advantages in terms of reducing the latency against [17] by the following reasons:

(1) Our proposed algorithm considers that the FUs with different bitwidths have different areas and different delays as well, and realizes operation chainings by calculating and utilizing such delays. Since our proposed algorithm realizes operation chainings on the critical paths including the interconnection delays, it is expected to reduce the control steps compared to [17].

(2) Our proposed algorithm appends some extra FUs into the vacant areas that help to realize more operation chainings, and it is expected to reduce more control steps.

4.6 Conclusion

In this chapter, we proposed a bitwidth-aware HLS algorithm with operation chainings for RDR architectures. Our proposed algorithm reduces the FU bitwidths and utilizes the vacant islands by adding extra FUs to realize effective operation chainings. Experimental results show that our algorithm reduces the latency by up to 47% compared to the algorithm with uniform bitwidth operation chainings. Our algorithm generates high performance circuits without increasing the total area.

However, the SA-based floorplanning remains the bottleneck in both speed and scalability as we discussed in Section 4.5.3. Accelerating the floorplanning is one of the future works. We do not evaluate the results on the real devices in this chapter. Another future work is to implement and evaluate our HLS solutions onto FPGA devices.

Chapter 5

A Fully-Connected Ising Model Embedding Method for 20k-Spin CMOS Annealing Machines[1]

5.1 Introduction

In Chapter 2, Chapter 3, and Chapter 4, floorplan-aware HLS algorithms are proposed. These algorithms successfully reduce the latency compared to the conventional algorithms while coping with the problem caused by increasing interconnection delays. However, the SA-based floorplanning in these algorithms remains the bottleneck in both speed and scalability. To deal with the bottleneck above, we try to apply and accelerate the floorplanning problem to the forthcoming Ising model-based computers (annealing machines). In this chapter, we firstly propose an embedding method of Ising models (which are equivalent to combinatorial optimization problems including the floorplanning problem) onto 20k-spin CMOS annealing machines (our targeting annealing machines).

Solving combinatorial optimization problems efficiently using novel physical annealing machines has been studied as in [7, 31, 46, 59, 69, 70, 72, 73]. In those studies, a combinatorial optimization problem is mapped onto a theoretical magnetic model in statistical mechanics called Ising model, and annealing machines search the ground-state of the Ising model, which corresponds to the optimal solution of the original combinatorial optimization problem. The overview of the Ising model and the annealing process are shown in Fig. 5.1. 20k-spin CMOS annealing machine [70] is one of the annealing machines and expected to be used to solve practical problems efficiently.

Figure 5.2 shows the overall flow of solving a combinatorial optimization problem using annealing machines. The flow consists of four phases. In Phase 1, the combina-

[1]Technical contents in this chapter have been presented in the publications ⟨5⟩ and ⟨6⟩.

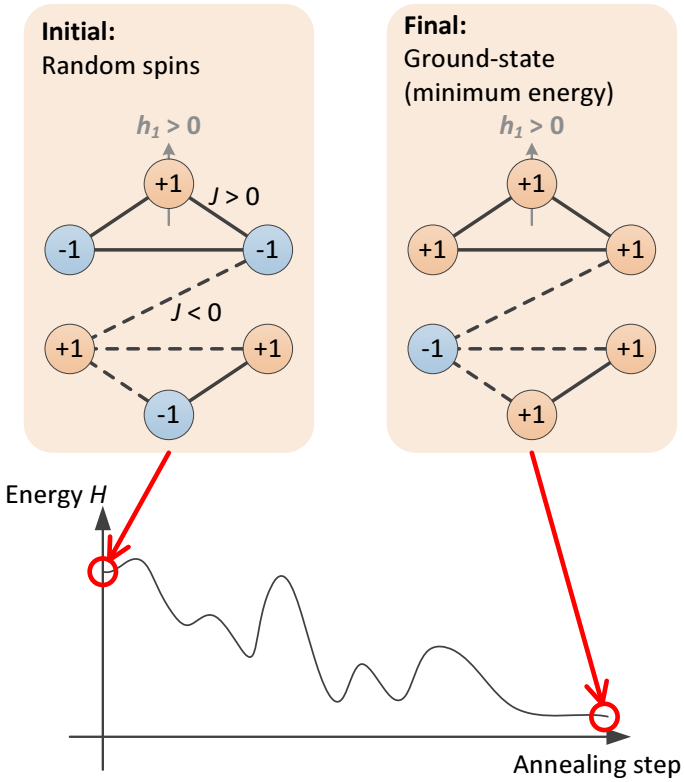


Figure 5.1: Overview of Ising model and annealing.

torial optimization problem to be solved is formulated (or mapped) onto an Ising model including the objective function and the constraints. In Phase 2, the Ising model formulated in Phase 1 is implemented on the physical annealing machine through Ising model embedding. This embedding determines the required spins, interactions between spins, and external magnetic fields of spins. In Phase 3, an annealing machine searches the ground-state (at the point of the minimum energy) of the spins. In Phase 4, by observing the final state of spins, we can obtain the solution of the input combinatorial optimization problem.

When we formulate combinatorial optimization problems by Ising models, the Ising models usually have interactions between any pairs of spins (fully-connected Ising models) because of the constraints of the problems. Generally, it is a hard task to embed a fully-connected Ising model onto the Ising model topology of a particular annealing machine. In this chapter, we propose a fully-connected Ising model embedding method for 20k-spin CMOS annealing machines [70]. The reason why we are only targeting fully-connected Ising models is that; (1) it is a typical topology of combinatorial optimization problems formulations as we described above; (2) once we embed a fully-connected Ising-model, we can embed any other Ising models, because any Ising models

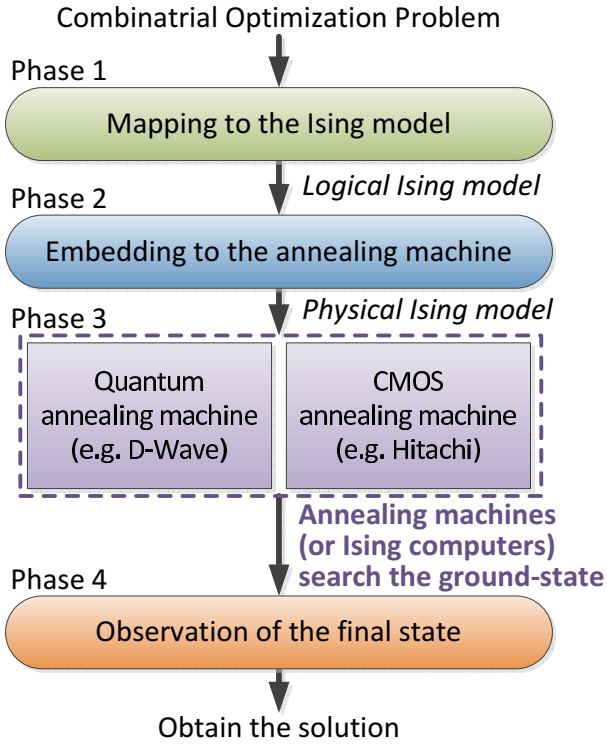


Figure 5.2: Overall flow of solving combinatorial optimization problems using annealing machines.

(equivalent to graphs) are sub-graphs of the fully-connected Ising model.

Some Ising model embedding methods have been proposed as in [7,8,61] for D-Wave quantum annealing machines (or D-Wave machines) [7,31], which is one of the annealing machines. However, since these methods are applicable only to the D-Wave machine topology or a general method, it is not efficient to apply these methods to 20k-spin CMOS annealing machines [70]. In this chapter, we propose an Ising model embedding method which maps a fully-connected Ising models that represent combinatorial optimization problems onto the Ising models on 20k-spin CMOS annealing machines [70], and evaluate our embedding method through solving practical combinatorial optimization problems.

The main contributions of this chapter are:

1. We propose *a general embedding method of Ising model* for 20k-spin CMOS annealing machine, which is expected to be one of the practical annealing machines, *for the first time* and keep the number of required physical spins to be $n^2 + n$ when the number of logical spins is n *theoretically*.

2. We prove that the ground state of the Ising models obtained from our proposed method is equivalent to that of the original Ising model.

3. Experimental results demonstrate that our proposed embedding method realizes generating Ising models with a fewer number of spins compared to the de facto standard conventional method in the practical problem size.

4. Experimental results through solving the practical combinatorial optimization problems demonstrate that our proposed embedding method can be superior in probabilities of feasible solutions and qualities of solutions compared to the conventional method.

The remainder of this chapter is organized as follows: Section 5.2 reviews conventional embedding methods for annealing machines as related works; Section 5.3 describes our targeting Ising model and topology of the annealing machine, and defines our embedding problem; Section 5.4 proposes our embedding method targeting the 20k-spin CMOS annealing machine; Section 5.5 shows experimental results and evaluates the efficiency of our method; Section 5.6 gives several concluding remarks.

5.2 Related Works

Many researches on solving combinatorial optimization problems have been done recently. Most of them are targeting D-Wave quantum annealing machine [7, 31]. D-Wave machine contains superconductor chips which work under very low temperature of milli-Kelvin order. The latest version of D-Wave machine has 2,048-spin (or -qubit) of Ising model. The Ising model topology on D-Wave machine is called a Chimera graph. The Chimera graph is a square-lattice of bipartite graphs $K_{4,4}$ topology.

A fully-connected Ising model embedding method has been proposed for the Ising model topology of D-Wave machine as in [7,61]. Bunyk *et al.* proposed a fully-connected Ising model embedding method onto the Chimera graph topology on D-Wave machines in [7]. In this method, the number of required spins on D-Wave machine is in proportion to the square of the number of spins in the input Ising model. Although method [7] can embed a fully-connected Ising model onto the Chimera graph topology on D-Wave machine, method [7] cannot be used for other annealing machines which have different Ising model topology from Chimera graphs. Thus, we cannot use method [7] for our targeting 20k-spin CMOS annealing machines.

An embedding method of an arbitrary topology of Ising model for D-Wave machine has also been proposed in [8]. Since such an embedding problem is known as NP-hard [24], Cai *et al.* proposed a heuristic algorithm based on the shortest-path search algorithm to solve the embedding problem in [8]. The method [8] is currently a de facto standard embedding method for D-Wave machines. Since method [8] is based on the general heuristic algorithm, both input Ising model topology and the targeting annealing

machine topology can be set by the user. Thus, we can use method [8] for our targeting 20k-spin CMOS annealing machines. However, since method [8] is a general heuristic, the embedding for 20k-spin CMOS annealing machines using this method is considered to be inefficient.

5.3 Problem Formulation

In this section, we define the Ising model, 20k-spin CMOS annealing machine, and our Ising model embedding problem.

5.3.1 Ising Model

Ising model is a theoretical magnetic model in statistical mechanics, which consists of microscopic variables called spins, interactions between them, and fields on each spin called external magnetic fields. An Ising model is represented as an undirected graph $\mathcal{M} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ and $\mathcal{E}$ are sets of spins and connections between spins, respectively. For each pair of spins $u \in \mathcal{V}$ and $v \in \mathcal{V}$ (where $u \neq v$), $(u, v) \in \mathcal{E}$ denotes a connection between spin u and v. Each spin $\sigma_i (i \in \mathcal{V})$ has either of the two values $+1/-1$ (or states up/down). The energy (or Hamiltonian) of an Ising model is calculated as

$$H = -\sum_{(i,j)\in\mathcal{E}} J_{ij}\sigma_i\sigma_j - \sum_{i\in\mathcal{V}} h_i\sigma_i \tag{5.1}$$

where J_{ij} is the interaction between spin σ_i and spin σ_j, h_i is the extra magnetic field of spin σ_i.

Logical Ising Model

In this chapter, we distinguish two types of Ising models; logical Ising models and physical Ising models.

A logical Ising model $M_L = (V_L, E_L)$ is an Ising model where interactions can be defined between any pairs of spins. Any graph topologies can be represented as logical Ising models. Combinatorial optimization problems are also formulated as and mapped to logical Ising models.

Spins on a logical Ising model are called logical spins. The interaction between logical spins σ_i and σ_j is denoted as $J_{L_{ij}}$, and the external magnetic field on a logical spin σ_i is denoted as h_{L_i}.

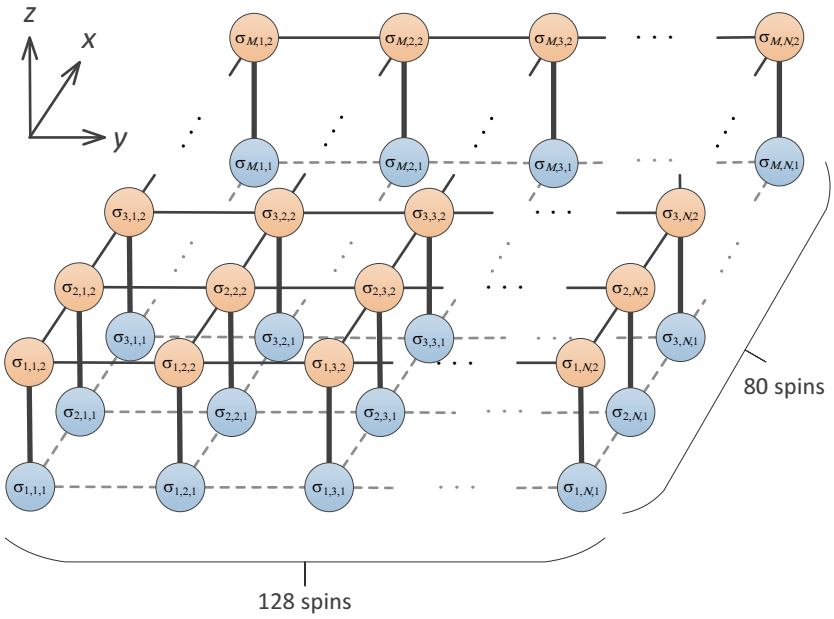


Figure 5.3: $128 \times 80 \times 2$-lattice Ising model topology of the 20k-spin CMOS annealing machine [70].

Physical Ising Model

A physical Ising model $M_P = (V_P, E_P)$ is an Ising model where interactions can only be defined between pairs of spins which are physically connected on the topology of individual annealing machine. If we have an annealing machine whose topology is fully-connected, we can directly and easily embed a logical Ising model which has interact connections between any pairs of spins onto the annealing machine. Generally, such annealing machines are not reasonable because it is complicated to connect spins which are located physically far away each others. Thus, typical annealing machines have physical Ising model topology where only physically adjacent spins are connected, for example. A physical Ising model M_P is a sub-graph of the Ising model topology of the target annealing machine. A physical Ising model can be easily embedded onto the annealing machine.

Spins on a physical Ising model are called physical spins. An interaction between physical spins σ_i and σ_j is denoted as $J_{P_{ij}}$, and an external magnetic field on a physical spin σ_i is denoted as h_{P_i}.

5.3.2 20k-Spin CMOS Annealing Machine [70]

In this chapter, our target annealing machine is 20k-spin CMOS annealing machine [70]. The overview of and the optimization in the 20k-spin CMOS annealing machine are

described below.

Overview of the Annealing Machine

The 20k-spin CMOS annealing machine is implemented by using CMOS technology (65-nm technology node) and was proposed by Yamaoka *et al.* in [70]. Since CMOS circuits are used to realize annealings of Ising models, 20k-spin CMOS annealing machine works at room temperature and does not need specific cooling equipment or power. Quantum annealing machines typically need them, on the other hand, 20k-spin CMOS annealing machine can be also designed and manufactured easily and have a better scalability by using conventional technologies compared to quantum annealing machines. The number of spins of 20k-spin CMOS annealing machine is 20,480 (20k).

Figure 5.3 shows $128 \times 80 \times 2$-lattice Ising model topology of 20k-spin CMOS annealing machine. The topology is a cube-lattice topology where every spin is connected to its adjacent spins. 20k-spin CMOS annealing machine has the two-layer of the square-lattice topology and each layer is composed of 128×80 physical spins. The axes are defined as shown in Fig. 5.3. A physical spin located at a coordinate (x, y, z) is denoted as $\sigma_{x,y,z}$. For example, the bottom-left spin in Fig. 5.3 is $\sigma_{1,1,1}$.

Optimization in the Annealing Machine

The optimization (annealing) in 20k-spin CMOS annealing machine [70] is described below. The details of the optimization are also described in [46]. In 20k-spin CMOS annealing machine, a value of each spin is updated to minimize the local energy calculated by the spin and its adjacent spins based on steepest descent method. To avoid sticking to local minima, the spin is inverted (or flipped) at the rate of "spin flipping probability." The "spin flipping probability" is given by the exponential function which depends on the current annealing step. The probability is defined as (a) "spin flipping probability (at the beginning)" when the step is zero and (b) "spin flipping probability (at the end)" when the step is the final step. These two probabilities are given as simulation parameters.

5.3.3 Embedding Logical Ising Models to Physical Ising Models

As we described in Section 5.3.1, we cannot usually embed a logical Ising model onto the annealing machine directly, while we can easily do a physical Ising model. To embed an Ising model onto the annealing machine, we need to convert a logical Ising model into an equivalent physical Ising model. To achieve this, we prepare several physical spins which represent a single logical spin. By preparing several redundant physical spins and mapping a single logical spins to one or more physical spins, every interaction between logical spins can be defined on the interaction between physical spins. These physical

spins which represent a single logical spin are called spin-chain. We can convert a logical Ising model into an equivalent physical Ising model using spin-chains and embed onto the annealing machine. Physical spins among a spin-chain are connected with the interaction of the strong ferromagnetic interactions ($J_F > 0$), so that every physical spins in a spin-chain will have the same value. Additionally, it is known that the uniform length of every spin-chain results in good result throughout the annealing process [6, 61]. The uniform length of spin-chains is desirable and we have to deal with this issue as well.

We define this embedding as an embedding problem of logical Ising models onto physical Ising models as follows:

Definition 5.1. *The embedding problem of a logical Ising model $M_L = (V_L, E_L)$ to a physical Ising model $M_P = (V_P, E_P)$ is, to define a mapping $\varphi : V_L \to 2^{V_P}$ which satisfies the following conditions:*

(Condition 1) *For all logical spins $\sigma_i \in V_L$, physical spins in $\varphi(\sigma_i)$ are connected,*

(Condition 2) *For all logical spins $\sigma_i \in V_L$ and $\sigma_j \in V_L$ (where $\sigma_i \neq \sigma_j$), $\varphi(\sigma_i) \cap \varphi(\sigma_j) = \varnothing$,*

(Condition 3) *If logical spins σ_i and σ_j are connected on M_L, one physical spin in $\varphi(\sigma_i)$ and one spin in $\varphi(\sigma_j)$ on M_P are connected.*

(Condition 4) *For all logical spins $\sigma_i \in V_L$ and $\sigma_j \in V_L$ (where $\sigma_i \neq \sigma_j$), $|\varphi(\sigma_i)| = |\varphi(\sigma_j)|$.*

□

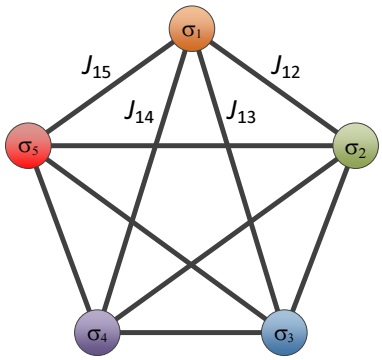
5.4 Proposed Embedding Method

In this section, we propose an embedding method which maps fully-connected logical Ising models to physical Ising models on 20k-spin CMOS annealing machine.

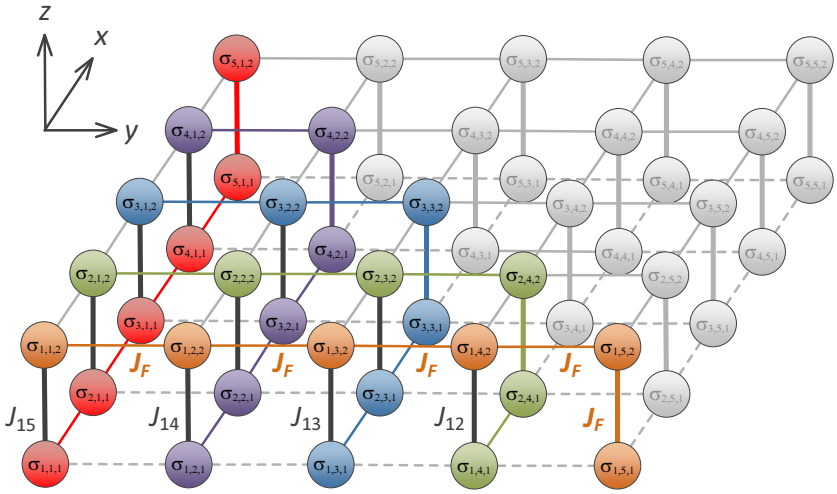
In a fully-connected topology, there must be connections between any pairs of logical spins. To express these connections onto the physical Ising model, we introduce the following ideas:

(Idea i) In the bottom layer ($z = 1$) of the 20k-spin CMOS annealing machine topology, physical spins which correspond to one logical spin are arranged vertically (x-axis direction) and connected with strong ferromagnetic interactions (J_F) each other to form a spin-chain.

(Idea ii) In the top layer ($z = 2$), such physical spins are arranged horizontally (y-axis direction) and connected with strong ferromagnetic interactions (J_F) each other to form a spin-chain.



(a) Logical Ising model.



(b) Physical Ising model (20k-spin CMOS Ising model).

Figure 5.4: Example of our proposed embedding method which maps (a) Input: Fully-connected logical Ising model (K_5) to (b) Output: Physical Ising model on 20k-spin CMOS annealing machine.

(Idea iii) Interactions J_{ij} are set to the corresponding connections between the bottom and the top layer.

Since all combinations of logical spins-pairs are appeared in the connections between the bottom and the top layer, any fully-connected logical Ising models can be expressed onto the physical Ising models.

Based on our ideas above, our proposed method is described below:

Input

A logical Ising model $M_L = (V_L, E_L)$ composed of n logical spins.

Output

A physical Ising model $M_P = (V_P, E_P)$ on 20k-spin CMOS annealing machine.

Step 1: Spins mapping

For each logical spin $\sigma_i \in V_L$, the mapping $\varphi(\sigma_i)$ is defined as follows:

$$\varphi(\sigma_i) = \bigcup_{x=1}^{i} \{\sigma_{x,n-i+1,1}\} \cup \bigcup_{y=1}^{n-i+1} \{\sigma_{i,y,2}\}. \tag{5.2}$$

Step 2: Interactions setting

Step 2.1

For each logical spin $\sigma_i \in V_L$, the interactions between the adjacent physical spins in $\varphi(\sigma_i)$ (obtained in Step 1) are set to J_F.

Step 2.2

For $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, n\}$ (where $i < j$), if there is a connection edge between the logical spin σ_i and σ_j, then the interaction between the physical spin $\sigma_{i'} = \sigma_{i,n-j+1,1}$ and $\sigma_{j'} = \sigma_{i,n-j+1,2}$ is set to be $J_{P_{i'j'}} = J_{L_{ij}}$. Otherwise, it is set to be $J_{P_{i'j'}} = 0$.

Step 2.3

For all connections on M_P whose interactions have not been set in Step 2.1 or Step 2.2, the interaction is set to be zero.

Step 3: External magnetic field setting

For each logical spin σ_i, to distribute the external magnetic field of σ_i onto the corresponding physical spins equivalently, the external magnetic fields of the physical spins $\sigma_{i'} \in \varphi(\sigma_i)$ ($\sigma_i \in V_L$) are set to be $h_{P_{i'}} = h_i/(n+1)$.

Example 5.1. *Figure 5.4 shows our Ising model embedding example of a fully-connected logical Ising model ($n = 5$) to a physical Ising model on 20k-spin CMOS annealing machine. In Fig. 5.4(b), physical spins colored by the same color correspond to one logical spin which has the same color in Fig. 5.4(a). For example, the logical spin σ_1 in Fig. 5.4(a) (orange-colored) corresponds to the physical spins $\{\sigma_{1,1,2}, \sigma_{1,2,2}, \sigma_{1,3,2}, \sigma_{1,4,2}, \sigma_{1,5,2}, \sigma_{1,5,1}\}$ in Fig. 5.4(b). The physical spins colored by gray in Fig. 5.4(b) are unused.* □

Theorem 5.1. *Our proposed embedding satisfies (Condition 1)–(Condition 4) in Definition 5.1.*

Proof. *Proof of satisfying (Condition 1): In Eq. (5.2) in Step 1, spins in the first term are connected along x-axis since it is a set of spins whose x-coordinate is increased by one*

from 1 *to* i*. Similarly, spins in the second term are connected along* y*-axis since it is a set of spins whose* y*-coordinate is increased by one from* 1 *to* $n-i+1$*. Furthermore, the physical spin where* $x = i$ *in the first them is* $\sigma_{i,n-i+1,1}$*, and the spin where* $y = n-i+1$ *in the second term is* $\sigma_{i,n-i+1,2}$*. These two physical spins are adjacent along* z*-axis. Thus, for all logical spins* $\sigma_i \in V_L$*, physical spins in* $\varphi(\sigma_i)$ *are connected.*

Proof of satisfying (Condition 2): In Eq. (5.2) in Step 1, y*-coordinate* $n-i+1$ *of the spins in the first term is a linear function depending on* i*. Similarly,* x*-coordinate* i *of the spins in the second term is also a linear function depending on* i*. This means that these coordinates must not be the same value when* i *is different.* z*-coordinate is always different between the first and the second terms of Eq. (5.2). Thus, physical spins do not overlap each other which means for all logical spins* $\sigma_i \in V_L$ *and* $\sigma_j \in V_L$ *(where* $\sigma_i \neq \sigma_j$*),* $\varphi(\sigma_i) \cap \varphi(\sigma_j) = \varnothing$*.*

Proof of satisfying (Condition 3): For all pairs of logical spins (σ_i, σ_j)*, every pair is connected in Step 2.2 one time. Thus, one physical spin in* $\varphi(\sigma_i)$ *and one spin in* $\varphi(\sigma_j)$ *on* M_P *are connected.*

Proof of satisfying (Condition 4): In Eq. (5.2) in Step 1, the number of physical spins in the first term is $(i-1+1)$*, and the number of physical spins in the second term is* $\{(n-i+1)-1+1\}$*. The number of total physical spins which corresponds to one logical spin is then calculated as* $(i-1+1)+\{(n-i+1)-1+1\} = n+1$*. For every logical spin, the number of physical spins which correspond to the logical spin is the same value* $(n+1)$*. Thus, for all logical spins* $\sigma_i \in V_L$ *and* $\sigma_j \in V_L$ *(where* $\sigma_i \neq \sigma_j$*),* $|\varphi(\sigma_i)| = |\varphi(\sigma_j)|$*.*

Therefore, our proposed embedding satisfies (Condition 1)–(Condition 4) in Definition 5.1. □

Our proposed method requires several redundant physical spins to embed logical Ising models onto physical Ising models. However, since physical spins in the same spin-chains are connected with strong ferromagnetic interactions (J_F) as we described in Section 5.3.3, the state of spins in the logical Ising model is equivalent to the state of spins in the physical Ising model. We define this as follows:

Definition 5.2. *If an embedding satisfies (Condition 1)–(Condition 3) of Definition 5.1, the ground state of* M_P *is equivalent to that of* M_L*.* □

Note that if an embedding does not satisfy (Condition 4), the ground state of the embedded Ising model is equivalent to that of the original Ising model.

Theorem 5.2. *The ground state of a physical Ising model obtained from our proposed embedding method is equivalent to that of the original logical Ising model.*

Proof. *Based on Theorem 5.1, out proposed method satisfies (Condition 1)–(Condition 3) of Definition 5.1. Therefore, Theorem 5.2 is proved.* □

Our proposed method maps one logical spin to several physical spins and connects them with strong ferromagnetic interactions (J_F) to form a spin-chain so that these spin values will become the same, as we described in Section 5.3.3. However, in a practical annealing process, all physical spins in the same spin-chain do not always have the same value. To take this problem into account, we determine the logical spin value by the majority vote of the corresponding physical spins. If the number of $+1$ and the number of -1 are the same, we randomly determine the corresponding logical spin value as $+1$ or -1. For example, assume that $\varphi(\sigma_1) = \{\sigma_a, \sigma_b, \sigma_c, \sigma_d, \sigma_e\}$. If $\sigma_a = \sigma_b = \sigma_c = \sigma_d = \sigma_e = +1$, then $\sigma_1 = +1$ obviously. If $\sigma_a = \sigma_b = +1$, $\sigma_c = \sigma_d = \sigma_e = -1$, then we determine $\sigma_1 = -1$.

The required physical spins when embedding fully-connected logical Ising model (K_n) by our proposed method is calculated as

$$(n + 1) \cdot n = n^2 + n. \tag{5.3}$$

The number of required spins is in proportion to the square of n (i.e. $O(n^2)$). Since the lattice topology of 20k-spin CMOS annealing machine is $128 \times 80 \times 2$, we can embed fully-connected Ising model (K_{80}) at the maximum by our proposed method.

5.5 Experimental Results

In this section, we evaluate the efficiency of our proposed method through some experiments.

5.5.1 Evaluation of Our Embedding Method

To evaluate the quality of our proposed embedding method, we compare the results of the de facto standard conventional embedding method [8] and our proposed method using fully-connected logical Ising models.

We have implemented the conventional method [8] and our proposed method in Python language on CentOS 6.8 and Intel Xeon CPU E5-2680 v3 2.50GHz $\times$ 40 machine with 270GB memory. For every fully-connected logical Ising model which has the number of logical spins of $n = 4$ to $n = 12$, we compare the embedding results obtained from the conventional method [8] and our proposed method targeting 20k-spin CMOS annealing machine. Since method [8] is based on the heuristic algorithm, the results may be different each time. We tried to perform method [8] for ten times. The comparison of the two embedding methods are shown in Table 5.1 and Fig. 5.5.

As seen in the result, we can conclude that our proposed method is superior to the conventional method [8] in the following three considerations:

Table 5.1: Comparison of embeddings of fully-connected logical Ising models (K_n) for the 20k-spin CMOS annealing machine. The "Min", the "Max", and the "Total spins" columns of "Method [8]" mean the minimum, the maximum, and the total lengths of the spin-chains (i.e. the number of physical spins which corresponds to one logical spin) of the best result (i.e. the result with the minimum total physical spins among we tried), respectively. The total numbers of physical spins obtained from our proposed method are $n^2 + n$ as we calculated in Section 5.4.

n	Method [8]			Our proposed method
	Min	Max	Total spins	Total spins
4	1	3	7	20
5	1	6	13	30
6	1	7	24	42
7	2	11	39	56
8	3	13	67	72
9	2	19	93	90
10	4	27	146	110
11	5	36	182	132
12	—	—	— *	156

* Computation of method [8] does not finish within one hour.

1. Method [8] requires fewer physical spins when n is small ($n \leq 8$). On the other hand, our proposed method requires fewer physical spins when n is large ($n \geq 9$), while [8] requires more physical spins. In practical problems, an Ising model with eight or less spins is considered to be too small, therefore, we can conclude that our proposed method requires fewer physical spins in the practical problem size compared to the conventional method [8].

2. Method [8] assigns logical spins to different lengths of physical spins (spin-chains) which may cause a worse annealing result, while our proposed method assigns every logical spins to the same lengths of physical spins (spin-chains), which may affect to a better annealing result.

3. Method [8] does not generate a solution when $n \geq 12$, while our proposed method can generate embedding solutions when n becomes large. Since our proposed method is based on the non-heuristic algorithm, we can obtain the stable embedding solutions easily.

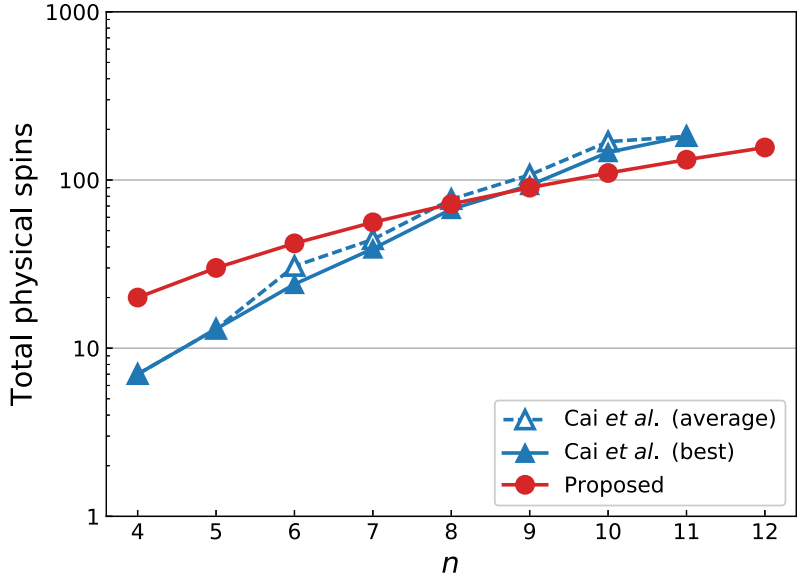


Figure 5.5: Comparison of required physical spins. The best of method [8] means the result with the minimum total physical spins among we tried. The average of method [8] means the average of all feasible solutions among we tried.

5.5.2 Evaluation of Our Method Applied to Combinatorial Optimizations

To evaluate our proposed embedding method in more practical uses, we solve MAX-CUT problem, which is one of the combinatorial optimization problems, using our proposed embedding method for the 20k-spin CMOS annealing machine.

Now, we solve the MAX-CUT problem of graph $G = (V, E)$ where V and E are sets of vertices and the edges of G, respectively. We aim to divide graph G into sub-graphs S and T. The objective is to divide G into two sub-graphs such that $|S| = |T|$ (when $|V|$ is even) or $|S| = |T| \pm 1$ (when $|V|$ is odd), while maximize the number of cut-edges (edges between S and T). This problem can be formulated as a logical Ising model. As in [37], we prepare $|V|$ spins of logical spins, each of which corresponds to each vertex in V. If σ is $+1$ then the corresponding vertex is considered to be in S, if σ is -1 then the corresponding vertex is considered to be in T. The Ising model of this problem is formulated as follows:

$$H = -\sum_{(i,j)\in E} \frac{1-\sigma_i\sigma_j}{2} + \alpha\left(\sum_{i\in V}\sigma_i\right)^2. \tag{5.4}$$

When the solution is optimal, Eq. (5.4) becomes minimum (ground-state). The first term

Table 5.2: Simulation parameters.

Parameter	Value
Initial spin value	Random
Spin flipping probability (at the beginning)	0.75
Spin flipping probability (at the end)	0.001
Annealing steps	100,000 **

** It takes $10x$ msec on a real machine (100 MHz) which inverts a $1/x$ part of total spins in one step. 20k-spin CMOS annealing machine has $x = 8$, thus the execution time would be 8 msec.

of Eq. (5.4) represents the objective function. For every edge, the first term reduces unity (if the edge is a cut-edge, i.e., $\sigma_i \neq \sigma_j$) or 0 (if the edge is not a cut-edge, i.e., $\sigma_i = \sigma_j = \pm 1$) from H. This means that when the number of cut-edges are the largest energy H is minimized. The second term of Eq. (5.4) represents the constraint. When $|S| = |T|$, the second term is zero. As the difference between $|S|$ and $|T|$ increases, the second term becomes large and it will be a penalty increasing the energy H. Here, we introduce a positive value α which is a weight parameter of the constraint term. Eq. (5.4) can be written as:

$$H = \frac{1}{2} \sum_{(i,j)\in E} \sigma_i \sigma_j + \alpha \sum_{i\in V, j\in V, i\neq j} \sigma_i \sigma_j + \text{const.} \tag{5.5}$$

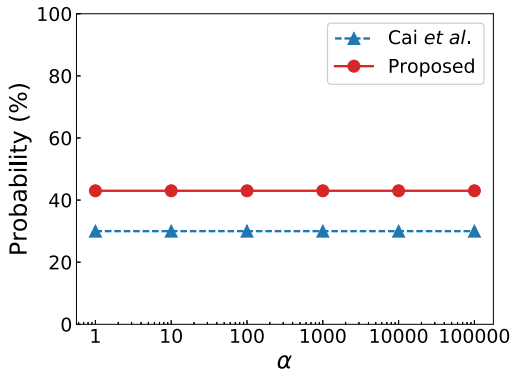
Comparing Eq. (5.5) to the energy of Ising model shown in Eq. (6.1), the interactions and the external magnetic fields of the Ising model are:

$$J_{ij} = \begin{cases} -\frac{1}{2} - \alpha & \text{if } (i,j) \in E \\ -\alpha & \text{otherwise} \end{cases} \tag{5.6}$$

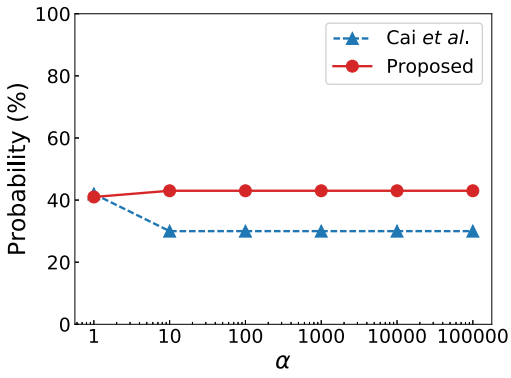
$$h_i = 0 \qquad \forall i \in V. \tag{5.7}$$

Comparison Between [8] and Our Method

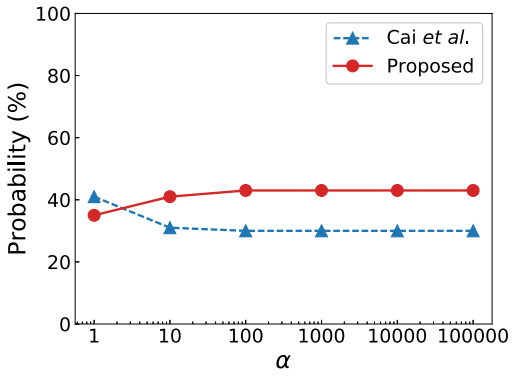
We solved the MAX-CUT problem using two embeddings obtained from [8] and ours using 20k-spin CMOS annealing machine simulator [70], to compare and evaluate these two embeddings. The benchmark graph SE3, which has the number of vertices $|V| = 8$, is picked up from Graph Collection [5] provided by Paderborn University AG-Monien. The parameters are set to be $\alpha \in \{1, 10, 100, 1000, 10000, 100000\}$ and $J_F \in \{0.1, 1, 10\}$. For each parameter, the simulation was performed 100 times. Simulation setting parameters are shown in Table 5.2. Spins randomly flip based on the "spin flipping probabilities (at the beginning/end)" to avoid sticking to local minima as described in Section 5.3.2.



(a) $J_F = 0.1$.



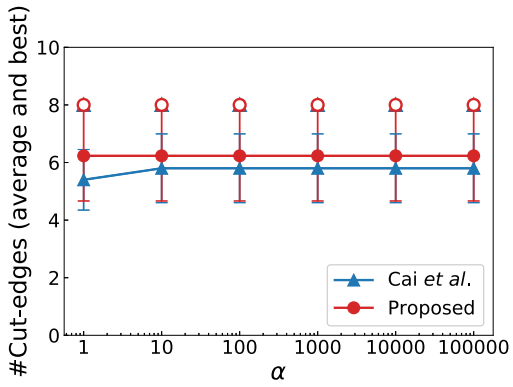
(b) $J_F = 1$.



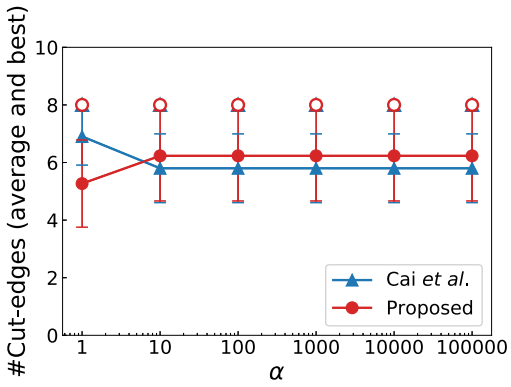
(c) $J_F = 10$.

Figure 5.6: Comparison of the probabilities of feasible solutions between [8] and our proposed method in the MAX-CUT problem (Graph: SE3).

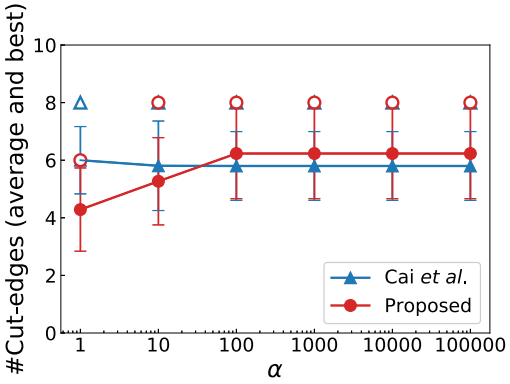
Figure 5.6 and Figure 5.7 show the results. The embedding used in method [8] is the embedding with the minimum physical spins at $n = 8$. Figure 5.6 and Figure 5.7 represent the probability of feasible solutions (solutions which satisfy the constraint of equally dividing) for each J_F and the quality of solution (the numbers of cut-edges) in feasible solution for each J_F, respectively. As seen in the result in Fig. 5.6, we can see that our proposed method has higher probabilities of feasible solutions except for the two cases of $(J_F, \alpha) = (1, 1)$ and $(J_F, \alpha) = (10, 1)$. To map every logical spin to the same length of physical spins by our method (i.e., to satisfy the *Condition 4* in Problem 5.1) is considered to be the reason for this result. As seen in the result in Fig. 5.7, we can see that our proposed method has better solution qualities on average when α is large.



(a) $J_F = 0.1$.



(b) $J_F = 1$.



(c) $J_F = 10$.

Figure 5.7: Comparison of the qualities of solutions (i.e. the numbers of cut-edges) between [8] and our proposed method in the MAX-CUT problem (Graph: SE3). The white markers represent the best solutions, the colors markers and lines represent the average solutions, and the error bars represent the standard deviations.

Evaluation of Our Method Using Other Graphs

We solved the MAX-CUT problem using our proposed embedding method using 20k-spin CMOS annealing machine simulator [70] for more benchmarks. 16 benchmark graphs are picked up from Graph Collection [5]. All these benchmark graphs have the number of vertices $|V| = 80$ or less, so that we can embedding to the 20k-spin CMOS annealing machine using our proposed method. The parameters are set to be $\alpha \in \{1, 10, 100, 1000\}$ and $J_F \in \{0.1, 1, 10\}$. For each parameter, the simulation was performed 100 times. The

simulation setting parameters are the same as in the experiment in Section 5.5.2.

Table 5.3, Table 5.4, and Table 5.5 show the results. We can see that the solution using our propose method can be obtained for FFT4 ($|V| = 80$) which has the largest spins embeddable onto the 20k-spin CMOS annealing machine. For most benchmark graphs in $\alpha = 1$ or $\alpha = 10$, when J_F becomes larger, the probability of feasible solutions tends to be increased while the quality of solutions tends to be decreased. For each J_F, when α becomes larger, the probability of feasible solutions tends to be decreased while the quality of solutions tends to be increased. We can see the above results in the MAX-CUT problem instances, general consideration about these parameters are one of the future works.

Table 5.3: Results of our proposed embedding method applied to MAX-CUT problem on the 20k-spin CMOS annealing machine (1/3).

Graph	α	#cut-edges***		
		$J_F = 0.1$	$J_F = 1$	$J_F = 10$
SE3 ($\|V\| = 8, \|E\| = 10$)	1	6.23 / 8 / 2 (43%)	5.27 / 8 / 2 (41%)	4.29 / 6 / 2 (35%)
	10	6.23 / 8 / 2 (43%)	6.23 / 8 / 2 (43%)	5.27 / 8 / 2 (41%)
	100	6.23 / 8 / 2 (43%)	6.23 / 8 / 2 (43%)	6.23 / 8 / 2 (43%)
	1000	6.23 / 8 / 2 (43%)	6.23 / 8 / 2 (43%)	6.23 / 8 / 2 (43%)
SE4 ($\|V\| = 16, \|E\| = 21$)	1	11.55 / 17 / 7 (31%)	8.33 / 12 / 3 (42%)	6.60 / 9 / 5 (70%)
	10	11.55 / 17 / 7 (31%)	11.55 / 17 / 7 (31%)	8.33 / 12 / 3 (42%)
	100	11.55 / 17 / 7 (31%)	11.55 / 17 / 7 (31%)	11.55 / 17 / 7 (31%)
	1000	11.55 / 17 / 7 (31%)	11.55 / 17 / 7 (31%)	11.55 / 17 / 7 (31%)
Grid4x4 ($\|V\| = 16, \|E\| = 24$)	1	13.55 / 21 / 7 (31%)	8.95 / 16 / 4 (42%)	5.49 / 10 / 4 (70%)
	10	13.55 / 21 / 7 (31%)	13.55 / 21 / 7 (31%)	8.95 / 16 / 4 (42%)
	100	13.55 / 21 / 7 (31%)	13.55 / 21 / 7 (31%)	13.55 / 21 / 7 (31%)
	1000	13.55 / 21 / 7 (31%)	13.55 / 21 / 7 (31%)	13.55 / 21 / 7 (31%)
CCA3 ($\|V\| = 24, \|E\| = 28$)	1	15.34 / 19 / 9 (32%)	7.52 / 15 / 4 (31%)	5.20 / 9 / 4 (51%)
	10	15.34 / 19 / 9 (32%)	15.34 / 19 / 9 (32%)	7.52 / 15 / 4 (31%)
	100	15.34 / 19 / 9 (32%)	15.34 / 19 / 9 (32%)	15.34 / 19 / 9 (32%)
	1000	15.34 / 19 / 9 (32%)	15.34 / 19 / 9 (32%)	15.34 / 19 / 9 (32%)
CCC3 ($\|V\| = 24, \|E\| = 36$)	1	17.44 / 22 / 12 (32%)	17.85 / 24 / 12 (26%)	16.94 / 18 / 16 (51%)
	10	17.44 / 22 / 12 (32%)	17.44 / 22 / 12 (32%)	17.85 / 24 / 12 (26%)
	100	17.44 / 22 / 12 (32%)	17.44 / 22 / 12 (32%)	17.44 / 22 / 12 (32%)
	1000	17.44 / 22 / 12 (32%)	17.44 / 22 / 12 (32%)	17.44 / 22 / 12 (32%)
BFLY3 ($\|V\| = 24, \|E\| = 48$)	1	25.31 / 32 / 18 (32%)	14.64 / 22 / 12 (28%)	10.08 / 14 / 8 (51%)
	10	25.31 / 32 / 18 (32%)	25.31 / 32 / 18 (32%)	14.64 / 22 / 12 (28%)
	100	25.31 / 32 / 18 (32%)	25.31 / 32 / 18 (32%)	25.31 / 32 / 18 (32%)
	1000	25.31 / 32 / 18 (32%)	25.31 / 32 / 18 (32%)	25.31 / 32 / 18 (32%)

*** In the "#cut-edges" column, each cell shows the number of cut-edges of feasible solutions in a format of "average / best / worst (probability of feasible solutions)."

Table 5.4: Results of our proposed embedding method applied to MAX-CUT problem on the 20k-spin CMOS annealing machine (2/3).

Graph	α	#cut-edges[***]		
		$J_F = 0.1$	$J_F = 1$	$J_F = 10$
SE5 ($\|V\| = 32, \|E\| = 46$)	1	24.40 / 32 / 18 (25%)	16.77 / 20 / 12 (26%)	15.32 / 16 / 14 (50%)
	10	24.40 / 32 / 18 (25%)	24.40 / 32 / 18 (25%)	16.77 / 20 / 12 (26%)
	100	24.40 / 32 / 18 (25%)	24.40 / 32 / 18 (25%)	24.40 / 32 / 18 (25%)
	1000	24.40 / 32 / 18 (25%)	24.40 / 32 / 18 (25%)	24.40 / 32 / 18 (25%)
FFT3 ($\|V\| = 32, \|E\| = 48$)	1	25.04 / 32 / 16 (25%)	11.85 / 22 / 8 (40%)	9.24 / 12 / 8 (50%)
	10	25.04 / 32 / 16 (25%)	25.04 / 32 / 16 (25%)	11.85 / 22 / 8 (40%)
	100	25.04 / 32 / 16 (25%)	25.04 / 32 / 16 (25%)	25.04 / 32 / 16 (25%)
	1000	25.04 / 32 / 16 (25%)	25.04 / 32 / 16 (25%)	25.04 / 32 / 16 (25%)
bcsstk01 ($\|V\| = 48, \|E\| = 176$)	1	91.41 / 100 / 78 (22%)	68.67 / 89 / 56 (21%)	60.37 / 65 / 58 (63%)
	10	91.41 / 100 / 78 (22%)	91.41 / 100 / 78 (22%)	68.67 / 89 / 56 (21%)
	100	91.41 / 100 / 78 (22%)	91.41 / 100 / 78 (22%)	91.41 / 100 / 78 (22%)
	1000	91.41 / 100 / 78 (22%)	91.41 / 100 / 78 (22%)	91.41 / 100 / 78 (22%)
CCA4 ($\|V\| = 64, \|E\| = 80$)	1	43.50 / 49 / 34 (20%)	10.76 / 17 / 8 (38%)	9.20 / 14 / 8 (59%)
	10	43.50 / 49 / 34 (20%)	43.50 / 49 / 34 (20%)	10.76 / 17 / 8 (38%)
	100	43.50 / 49 / 34 (20%)	43.50 / 49 / 34 (20%)	43.50 / 49 / 34 (20%)
	1000	43.50 / 49 / 34 (20%)	43.50 / 49 / 34 (20%)	43.50 / 49 / 34 (20%)
SE6 ($\|V\| = 64, \|E\| = 93$)	1	48.10 / 60 / 40 (20%)	32.65 / 37 / 31 (17%)	30.76 / 33 / 29 (59%)
	10	48.10 / 60 / 40 (20%)	48.10 / 60 / 40 (20%)	32.65 / 37 / 31 (17%)
	100	48.10 / 60 / 40 (20%)	48.10 / 60 / 40 (20%)	48.10 / 60 / 40 (20%)
	1000	48.10 / 60 / 40 (20%)	48.10 / 60 / 40 (20%)	48.10 / 60 / 40 (20%)

[***] In the "#cut-edges" column, each cell shows the number of cut-edges of feasible solutions in a format of "average / best / worst (probability of feasible solutions)."

Table 5.5: Results of our proposed embedding method applied to MAX-CUT problem on the 20k-spin CMOS annealing machine (3/3).

Graph	α	#cut-edges***		
		$J_F = 0.1$	$J_F = 1$	$J_F = 10$
CCC4 ($\|V\| = 64, \|E\| = 96$)	1	48.10 / 62 / 40 (20%)	34.77 / 36 / 34 (13%)	32.71 / 34 / 32 (59%)
	10	48.10 / 62 / 40 (20%)	48.10 / 62 / 40 (20%)	34.77 / 36 / 34 (13%)
	100	48.10 / 62 / 40 (20%)	48.10 / 62 / 40 (20%)	48.10 / 62 / 40 (20%)
	1000	48.10 / 62 / 40 (20%)	48.10 / 62 / 40 (20%)	48.10 / 62 / 40 (20%)
Grid8x8 ($\|V\| = 64, \|E\| = 112$)	1	57.40 / 76 / 47 (20%)	12.67 / 18 / 8 (24%)	8.97 / 12 / 8 (59%)
	10	57.40 / 76 / 47 (20%)	57.40 / 76 / 47 (20%)	12.67 / 18 / 8 (24%)
	100	57.40 / 76 / 47 (20%)	57.40 / 76 / 47 (20%)	57.40 / 76 / 47 (20%)
	1000	57.40 / 76 / 47 (20%)	57.40 / 76 / 47 (20%)	57.40 / 76 / 47 (20%)
BFLY4 ($\|V\| = 64, \|E\| = 128$)	1	66.90 / 74 / 60 (20%)	19.79 / 28 / 16 (39%)	17.93 / 24 / 16 (59%)
	10	66.90 / 74 / 60 (20%)	66.90 / 74 / 60 (20%)	19.79 / 28 / 16 (39%)
	100	66.90 / 74 / 60 (20%)	66.90 / 74 / 60 (20%)	66.90 / 74 / 60 (20%)
	1000	66.90 / 74 / 60 (20%)	66.90 / 74 / 60 (20%)	66.90 / 74 / 60 (20%)
bcsstk02 ($\|V\| = 66, \|E\| = 2145$)	1	1089.00 / 1089 / 1089 (13%)	1089.00 / 1089 / 1089 (13%)	1089.00 / 1089 / 1089 (52%)
	10	1089.00 / 1089 / 1089 (13%)	1089.00 / 1089 / 1089 (13%)	1089.00 / 1089 / 1089 (13%)
	100	1089.00 / 1089 / 1089 (13%)	1089.00 / 1089 / 1089 (13%)	1089.00 / 1089 / 1089 (13%)
	1000	1089.00 / 1089 / 1089 (13%)	1089.00 / 1089 / 1089 (13%)	1089.00 / 1089 / 1089 (13%)
FFT4 ($\|V\| = 80, \|E\| = 128$)	1	65.10 / 78 / 56 (20%)	20.50 / 36 / 16 (32%)	17.30 / 22 / 16 (57%)
	10	65.10 / 78 / 56 (20%)	65.10 / 78 / 56 (20%)	20.50 / 36 / 16 (32%)
	100	65.10 / 78 / 56 (20%)	65.10 / 78 / 56 (20%)	65.10 / 78 / 56 (20%)
	1000	65.10 / 78 / 56 (20%)	65.10 / 78 / 56 (20%)	65.10 / 78 / 56 (20%)

*** In the "#cut-edges" column, each cell shows the number of cut-edges of feasible solutions in a format of "average / best / worst (probability of feasible solutions)."

5.6 Conclusion

In this chapter, we proposed a fully-connected Ising model embedding method for 20k-spin CMOS annealing machines. Experimental results effectively show that our proposed method embeds Ising models using less physical spins compared to the conventional de facto standard method in the practical problem size, and that the probability of feasible solutions and the solution quality using our proposed method is better than those of the conventional method when solving practical combinatorial optimization problems.

General considerations and optimizations of the annealing parameters, and applying our method to other combinatorial optimization problems are future works.

Chapter 6

Rectangle Packing by Ising Computers

6.1 Introduction

In Chapter 5, we firstly propose an embedding method of Ising models onto 20k-spin CMOS annealing machines. In this chapter, to accelerate the floorplanning problem in Chapter 2, Chapter 3, and Chapter 4, which is the bottleneck in both speed and scalability as described in Section 4.5.3, a floorplanning problem is mapped onto an Ising model. Annealings of the mapped Ising model are performed by using the embedding method proposed in Chapter 5.

Floorplanning of modules has been a significant role in VLSI design automation. Floorplanning can be formulated as the "Rectangle Packing Problem" (as shown in Fig. 6.1[1]) and many floorplanning methods have been proposed in this couple of decades. It is known that floorplanning techniques are applied for not only VLSI design automation but also other domains such as the strip-packing problem (or marker making problem) [22] and the nesting problem [50] which are related to design methodologies in the textile industry.

Ising model-based computers (or annealing machines) are the type of a non-von Neumann computer and recently studied and expected to solve combinatorial optimization problems efficiently. Annealing machines search the ground-state of Ising models. If we give an assignment for the optimal solution of a combinatorial optimization problem to the ground-state of an Ising model (i.e., mapping of a problem onto an Ising model), we can solve the combinatorial optimization problem by annealing machines. Some physical annealing machines have been developed in [7, 31, 59, 70, 73]. D-Wave quantum annealing machine [7, 31] has a superconductor chips which works under very low temperature. D-Wave machine has 2,048 spins (or qubits) and they are connected in Chimera graph topology. CMOS annealing machines [70, 73] are annealing machines

[1]The problem details are described in Section 6.2.

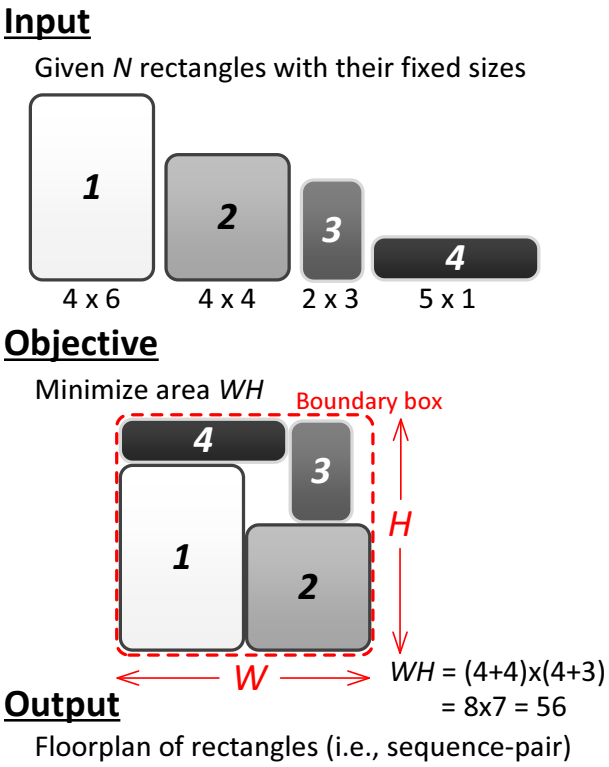


Figure 6.1: An example of "Rectangle Packing Problem."

using CMOS circuits. 20k-spin CMOS annealing machine [70] is implemented using CMOS (65-nm) technology which works at room temperature. It has 20,480 (20k) spins and they are connected in $128 \times 80 \times 2$-lattice topology. According to [70], the power efficiency of 20k-spin CMOS annealing machine is 1,800 times higher compared to that of a 1.87-GHz Intel Core i5 processor. FPGA-based annealing machine by Fujitsu Laboratories Ltd. [59] has 1,024 spins which are fully connected. According to [59], it can solve a combinatorial optimization problem 12,000 times faster compared to a 3.5-GHz Intel Xeon E5-1620 v3 processor.

Solving relatively simple problems by annealing machines are studied so far. However, not so many works focus on the more practical problems. The scheduling problems, which are ones of relatively practical problems and related to EDA domains, are solved on the D-Wave machine as a problem of "Job-Shop Scheduling" in [62]. Regarding the "Rectangle Packing Problem," which is also one of relatively practical problem, sequence-pair [44] has a remarkable contribution. Sequence-pair can represent any floorplans of rectangles by using two sequences of rectangles. Sequence-pair itself is a representation of a floorplan. We generally optimize a sequence-pair (or floorplan) using meta heuristic methods such as simulated annealing (SA) [32], where the computation time may be the bottleneck. Thus, it is worth applying Ising model-based computations

for it. For the "Rectangle Packing Problem," the energy function of Ising model has been briefly proposed in [68], recently. They firstly proposed a mapping of "Rectangle Packing Problem" to an Ising model and constructed the energy function. However, they only evaluated the energy and did not evaluate the solutions of the problem. They does not evaluate the performance with keeping the use of annealing machine in mind as well.

In this chapter, we try to solve the floorplanning problem by annealing machines to investigate possibilities of annealing machines. We propose a mapping of "Rectangle Packing Problem" for solving it by the annealing machines. In our proposed mapping, a sequence-pair is represented on an Ising model, and the energy function to obtain the optimal solution of the problem is constructed. Our proposed approach maps a "Rectangle Packing Problem" with N rectangles onto a $3N^3$-spin logical Ising model. Experimental results demonstrate that through the proposed approach we can solve the problem with nine rectangles at the maximum on a fully-connected annealing machine and the problem with three rectangles at the maximum on 20k-spin CMOS annealing machine.

The contributions of this chapter are:

1. We realize the Ising model mapping of "Rectangle Packing Problem" with N rectangles onto the $3N^3$-spin Ising model using sequence-pair for solving it by the Ising model-based computers.

2. Experimental results successfully demonstrate that through the proposed approach we can solve the problem with nine rectangles at the maximum on a fully-connected annealing machine and the problem with three rectangles at the maximum on 20k-spin CMOS annealing machine.

This chapter is organized as follows: Section 6.2 describes our problem formulation; Section 6.3 proposes our Ising model mapping to solve the "Rectangle Packing Problem"; Section 6.4 shows experimental results and discussion; Section 6.5 gives several concluding remarks and future works.

6.2 Problem Definition

Figure 6.1 shows an example of "Rectangle Packing Problem." Assume that a set of N rectangles $R = \{r_i | 1 \leq i \leq N\}$ are given. Let w_i and h_i be the width and height of a rectangle r_i, respectively. In this chapter, the width and height of a rectangle is fixed, and no rotations are allowed, for simplicity. Let W and H be the width and the height of the boundary box, respectively. The objective is to minimize the boundary box area WH without overlapping of rectangles. The "Rectangle Packing Problem" is defined as follows:

Definition 6.1. *For a given R (a set of N rectangles with their width and height), the "Rectangle Packing Problem" is, to minimize WH and generate its floorplan without overlapping rectangles where W and H are the width and the height of the boundary box, respectively.* □

This problem is known as an NP-hard problem [44].

6.3 Proposed Ising Model Mapping of Rectangle Packing Problem

6.3.1 Sequence-Pair [44]

A few decades ago, Murata *et al.* proposed a remarkable floorplan representation named sequence-pair [44]. The main advantage of sequence-pair is that any floorplans of rectangles can be represented by two sequences of rectangles. Since sequence-pair itself is a representation of a floorplan, we generally optimize a sequence-pair (or floorplan) using metaheuristic methods such as simulated annealing (SA) [32]. For example, a representation by sequence-pair of Fig. 6.1 is written as (Γ_+, Γ_-) = (4 1 3 2, 1 2 4 3).

From a sequence-pair to a floorplan of rectangles, they proposed an $O(N^2)$ algorithm by generating the oblique grid and horizontal/vertical constraint graphs (G_h/G_v) followed by finding longest paths in G_h and G_v in [44].

6.3.2 Ising Model

Ising model is a theoretical magnetic model in statistical mechanics, which consists of spins, interactions between spins, and external magnetic fields on spins. An Ising model is defined on an undirected graph $M = (V, E)$ where V and E are sets of spins and interactions between spins, respectively. For each pair of spins $u \in V$ and $v \in V$ (where $u \neq v$), $(u, v) \in E$ denotes an interaction between spin u and v. Each spin s_i ($i \in V$) has either of the two values $+1/-1$ (or states up/down). The energy function (or Hamiltonian) of an Ising model is given by

$$\mathcal{H} = -\sum_{(i,j)\in E} J_{ij} s_i s_j - \sum_{i\in V} h_i s_i, \tag{6.1}$$

where J_{ij} is the interaction between spin s_i and spin s_j, h_i is the external magnetic field of spin s_i.

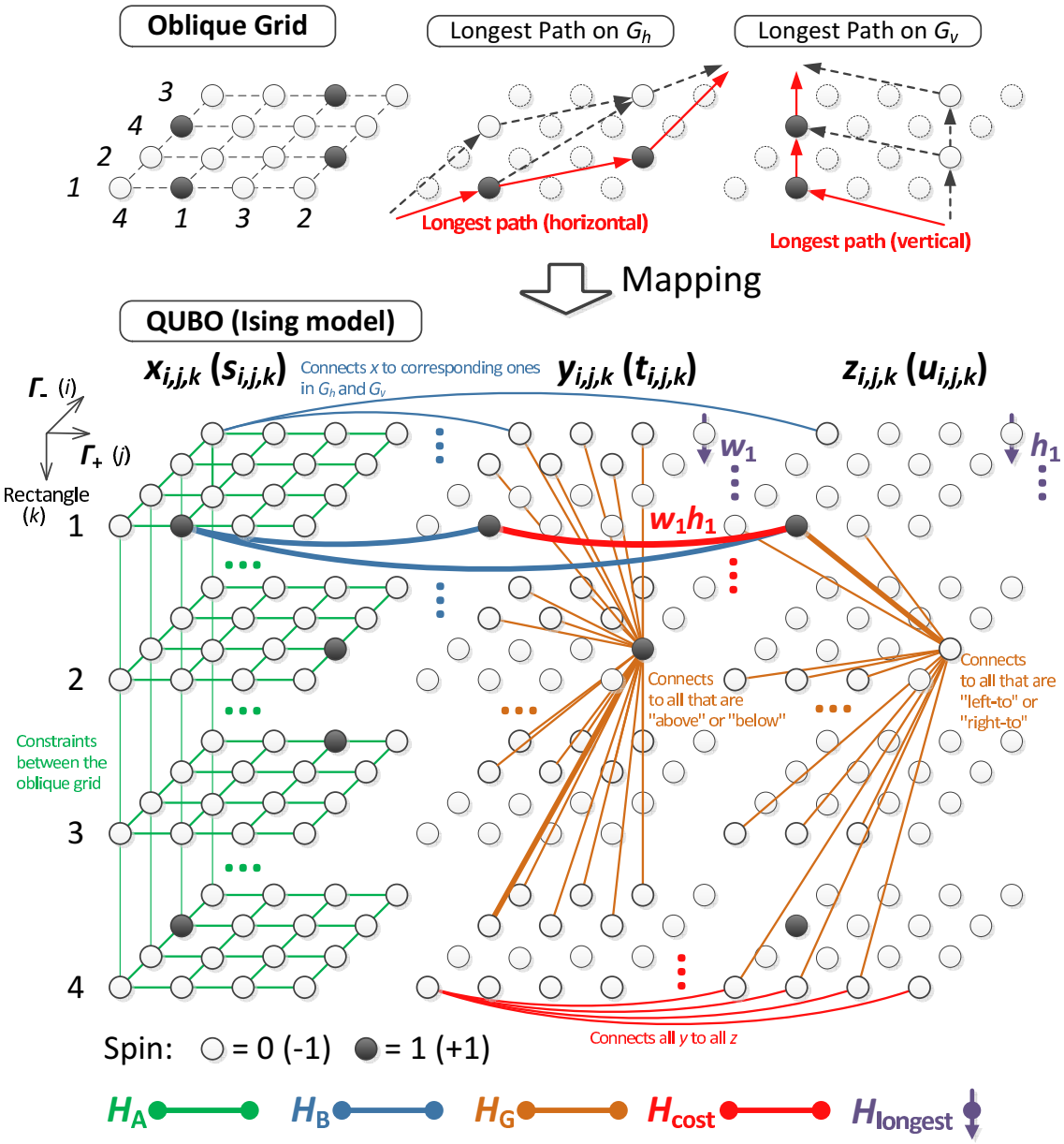


Figure 6.2: Our proposed mapping to an Ising model for the "Rectangle Packing Problem." When the number of rectangles is N, we need three parts of N^3-spin Ising models. In total, $3N^3$ logical spins are required. The colored edges represent interactions between spins, but many of them are omitted.

6.3.3 Mapping to Ising Model and Energy Function

Figure 6.2 shows our proposed Ising model mapping. Our approach maps "Rectangle Packing Problem" using sequence-pair to an Ising model. The whole Ising model consists of three parts: (1) oblique grid, (2) horizontal constraint graph (G_h), and (3) vertical constraint graph (G_v). Similar to [68], each of them is composed of N^3 spins.

Let $x_{i,j,k}$, $y_{i,j,k}$, and $z_{i,j,k}$ be 0/1 (zero or one) binary variables. Introducing 0/1 binary variables makes us easy to construct the energy function. The energy function using 0/1 binary variables is called a QUBO (Quadratic Unconstrained Binary Optimization) representation.

Let us introduce the details of these three variables:

$x_{i,j,k}$: In the oblique grid, $x_{i,j,k} = 1$ means that rectangle r_k is at the i-th order in Γ_+ and at the j-th order in Γ_-.

$y_{i,j,k}$: In the horizontal constraint graph (G_h), $y_{i,j,k} = 1$ means that it is on the longest path on G_h. $y_{i,j,k} = 0$ means that it is not.

$z_{i,j,k}$: In the vertical constraint graph (G_v), $z_{i,j,k} = 1$ means that it is on the longest path on G_v. $z_{i,j,k} = 0$ means that it is not.

To map a QUBO to an Ising model, we need to convert binary variables (0/1) to spin variables ($-1/+1$). Let $s_{i,j,k}$, $t_{i,j,k}$, and $u_{i,j,k}$ be spin variables corresponding to the binary variables $x_{i,j,k}$, $y_{i,j,k}$, and $z_{i,j,k}$, respectively. A 0/1 binary values $x_{i,j,k}$ can be converted to a $-1/+1$ spin variable by the following equation:

$$x_{i,j,k} = \frac{s_{i,j,k} + 1}{2} \tag{6.2}$$

Similarly, $y_{i,j,k}$ and $z_{i,j,k}$ can be converted to $t_{i,j,k}$ and $u_{i,j,k}$, respectively.

Necessary Constraint: In Oblique Grid

In the oblique grid, every rectangle must appear only once. In other words, in our Ising model mapping, the following constraint must be satisfied for all k:

$$\sum_{i=1}^{N} \sum_{j=1}^{N} x_{i,j,k} = 1 \quad (\forall k). \tag{6.3}$$

To express this constraint in the energy function of Ising model, we need to minimize the following terms:

$$\begin{aligned}
&\left(\sum_{i=1}^{N}\sum_{j=1}^{N} x_{i,j,k} - 1\right)^2 \\
=\ &\left(\sum_{i=1}^{N}\sum_{j=1}^{N} \frac{s_{i,j,k}+1}{2} - 1\right)^2 \\
=\ &\left\{\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \left(s_{i,j,k}+1\right) - 1\right\}^2 \\
=\ &\left(\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} s_{i,j,k} + \frac{N^2}{2} - 1\right)^2 \\
=\ &\frac{1}{4}\left(\sum_{i=1}^{N}\sum_{j=1}^{N} s_{i,j,k}\right)^2 + \frac{N^2-2}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} s_{i,j,k} + \left(\frac{N^2-2}{2}\right)^2 \\
=\ &\frac{1}{4}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{i'=1}^{N}\sum_{j'=1}^{N} s_{i,j,k}s_{i',j',k} + \frac{N^2-2}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} s_{i,j,k} + \text{const.}
\end{aligned} \tag{6.4}$$

There must also exist only one rectangle for each horizontal/vertical oblique line. Therefore, the following constraints must be satisfied for all i and for all j like Eq. (6.3):

$$\sum_{j=1}^{N}\sum_{k=1}^{N} x_{i,j,k} = 1 \quad (\forall i), \tag{6.5}$$

$$\sum_{i=1}^{N}\sum_{k=1}^{N} x_{i,j,k} = 1 \quad (\forall j). \tag{6.6}$$

Similarly to Eq. (6.4), we convert these $x_{i,j,k}$ to $s_{i,j,k}$.

By defining constraints for all i, j, and k, we can construct $\mathcal{H}_A$. In Fig. 6.2, a part of interactions related to $\mathcal{H}_A$ are shown in green color.

These three constraints are necessary constraints. If any of these constraints are not satisfied, the floorplan of rectangles and the area are undefined, since the sequence-pair cannot be generated.

Supplementary Constraint 1: Between Oblique Grid and Constraint Graphs

In the Ising model representing the horizontal constraint graphs (G_h), let each spin $y_{i,j,k}$ correspond to $x_{i,j,k}$. If $x_{i,j,k} = 0$, then $y_{i,j,k}$ should be zero. If $x_{i,j,k} = 1$, then $y_{i,j,k}$ should

be one when $y_{i,j,k}$ is on the longest path of G_h, or zero when $y_{i,j,k}$ is not on the longest path of G_h. In this constraint, we just express duplications of $x_{i,j,k}$ as $y_{i,j,k}$ by putting a positive interaction between two spin variables representing the oblique grid and the spin representing the horizontal constraint graph. We set the same interactions for the vertical constraint graph. The energy function is:

$$\mathcal{H}_B = -\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N} s_{i,j,k}t_{i,j,k} - \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N} s_{i,j,k}u_{i,j,k}. \tag{6.7}$$

The first term in the right-hand side of Eq. (6.7) is minimized, when the sign of $s_{i,j,k}$ and the sign of $t_{i,j,k}$ is the same (i.e., $s_{i,j,k} = t_{i,j,k} = 1$ or $s_{i,j,k} = t_{i,j,k} = -1$). When the signs are different, the energy increases. We can say similar way for the second term.

We can construct $\mathcal{H}_B$ as described above. In Fig. 6.2, a part of interactions related to $\mathcal{H}_B$ are shown in blue color.

Note that this constraint is a supplementary constraint, which means that even if this constraint is not satisfied, the solution is feasible as long as the "Necessary Constraint" described in Section 6.3.3 is satisfied.

Supplementary Constraint 2: In Constraint Graphs

For each constraint graphs, we introduce constraints which prohibit spins to be one simultaneously.

For the horizontal constraint graph, for each $r \in R$, any rectangles which are relatively "above" or "below" of r must not be selected as the longest path simultaneously. To represent this constraint in the Ising model, we construct the following energy function:

$$\mathcal{H}_G^{\text{horizontal}} = \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N}\sum_{y'\in A_{i,j,k}} y_{i,j,k}\,y' \tag{6.8}$$

where $A_{i,j,k}$ is the set of spins which are "above" on the G_h from a spin at the position of (i, j, k). Similarly for G_v, $\mathcal{H}_G^{\text{vertical}}$ is constructed.

The sum of these two energy function $\mathcal{H}_G = \mathcal{H}_G^{\text{horizontal}} + \mathcal{H}_G^{\text{vertical}}$ is the energy function for "Supplementary Constraint 2." In Fig. 6.2, a part of interactions related to $\mathcal{H}_G$ are shown in orange color.

Note that this constraint is a supplementary constraint as well as "Supplementary Constraint 1."

Objective Function 1: Area

The following energy function is minimized when the area of the boundary box (WH) is minimum:

$$\begin{aligned}\mathcal{H}_{\text{area}} &= WH \\ &= \left(\sum_{i\in P_h} w_i\right)\left(\sum_{j\in P_v} h_j\right) \\ &= \left(\sum_{i=1}^{N} w_i y_i\right)\left(\sum_{j=1}^{N} h_j z_j\right) \\ &= \sum_{i=1}^{N}\sum_{j=1}^{N}(w_i h_j) y_i z_j, \end{aligned} \tag{6.9}$$

where P_h and P_v is a set of rectangles on one of the paths in G_h and G_v, respectively.

Objective Function 2: Longest Path

The energy function introduced in "Objective Function 1: Area" does not ensure that the longest path is selected in G_h and G_v. The following energy function is minimized when the longest path is selected in both G_h and G_v.

$$\mathcal{H}_{\text{longest}} = \sum_{i}^{N}(-w_i y_i) + \sum_{i}^{N}(-h_i z_i)\,. \tag{6.10}$$

Overall Energy Function

The overall energy function is the weighted sum of all energy functions calculated above:

$$\mathcal{H} = \alpha\mathcal{H}_A + \beta\mathcal{H}_B + \gamma\mathcal{H}_G + \delta\mathcal{H}_{\text{area}} + \epsilon\mathcal{H}_{\text{longest}} \tag{6.11}$$

where α, β, γ, δ, and ϵ are the user-specified (or hyper) parameters.

6.4 Experiments and Discussion

In this section, we evaluate our proposed mapping by using the Ising model simulator.

6.4.1 Experimental Results

Table 6.1 shows the summary of benchmarks used in our experiments. Note that "inst4" is the same as in Fig. 6.1.

Table 6.1: Summary of benchmarks.

Name	N	Size of each rectangles
inst1	3	$\{1 \times 1, 1 \times 1, 1 \times 1\}$
inst2	3	$\{4 \times 6, 4 \times 4, 2 \times 3\}$
inst3	3	$\{40 \times 60, 40 \times 40, 20 \times 30\}$
inst4	4	$\{4 \times 6, 4 \times 4, 2 \times 3, 5 \times 1\}$
inst5	5	$\{4 \times 6, 4 \times 4, 2 \times 3, 5 \times 1, 1 \times 1\}$
inst6	6	$\{4 \times 6, 4 \times 4, 2 \times 3, 5 \times 1, 1 \times 1, 7 \times 8\}$
inst7	7	$\{4 \times 6, 4 \times 4, 2 \times 3, 5 \times 1, 1 \times 1, 7 \times 8, 10 \times 2\}$
inst8	8	$\{4 \times 6, 4 \times 4, 2 \times 3, 5 \times 1, 1 \times 1, 7 \times 8, 10 \times 2, 3 \times 9\}$
inst9	9	$\{4 \times 6, 4 \times 4, 2 \times 3, 5 \times 1, 1 \times 1, 7 \times 8, 10 \times 2, 3 \times 9, 5 \times 7\}$
inst10	10	$\{4 \times 6, 4 \times 4, 2 \times 3, 5 \times 1, 1 \times 1, 7 \times 8, 10 \times 2, 3 \times 9, 5 \times 7, 4 \times 1\}$

The width and height of each rectangle are normalized since the amount of area causes an influence to the energy value of $\mathcal{H}_{\text{area}}$ (calculated in Eq. (6.9)) and that of $\mathcal{H}_{\text{longest}}$ (calculated in Eq. (6.10)) if they are not. For example, if the width and height are too small, $\mathcal{H}_{\text{area}}$ tends to be smaller than the other energy costs. If the width and height are too large, $\mathcal{H}_{\text{area}}$ tends to be larger than the other energy costs. We need to remove this influence. Let W_{sum} and H_{sum} be the sum of the width and the height of all the rectangles, respectively, w_i and h_i for all i are normalized as follows:

$$\begin{cases} w_i' & \leftarrow & \dfrac{w_i}{W_{\text{sum}} + H_{\text{sum}}}, \\ h_i' & \leftarrow & \dfrac{h_i}{W_{\text{sum}} + H_{\text{sum}}}. \end{cases} \tag{6.12}$$

To check the effects for this normalization, we prepare "inst2" and "inst3" as in Table 6.1.

We have implemented our proposed approach by Python and C++. The Ising model simulator by Hitachi, Ltd. [70] was used. We tried two types of annealing machine as:

Complete: For an annealing machine which is assumed to have an Ising model topology of a complete graph. [59] is one of such annealing machines.

[70] w/ Ch. 5: For 20k-spin CMOS annealing machine [70] with a modified version of the embedding method proposed in Chapter 5. The proposed method in Chapter 5 determines the logical spin value by the majority vote of all the corresponding physical spins. In this chapter for "Rectangle Packing Problem," we determine the logical spin value by the majority vote of the corresponding physical spins which are connected to spins representing the oblique grid, since only spins for the oblique grid is evaluated while the other spins are for the area calculation. We set the strong ferromagnetic interaction value as $J_F = 4$.

Table 6.2: Experimental results.

Name	N	Target	#spins	Prob. (%)	Area (best)	Area (avg.)
inst1	3	Complete	81	42.0	3	3.62 ± 0.49
		[70] w/ Ch. 5	6642	0.2	4	4.00*
inst2	3	Complete	81	52.0	52	61.00 ± 6.56
		[70] w/ Ch. 5	6642	0.2	60	60.00*
inst3	3	Complete	81	52.0	5200	6100.00 ± 656.07
		[70] w/ Ch. 5	6642	0.2	6000	6000.00*
inst4	4	Complete	192	54.0	56	85.37 ± 17.57
		[70] w/ Ch. 5	37056	0	—	—
inst5	5	Complete	375	34.4	56	91.95 ± 18.35
		[70] w/ Ch. 5	141000	—	—	—
inst6	6	Complete	648	25.4	128	186.72 ± 34.97
		[70] w/ Ch. 5	420552	—	—	—
inst7	7	Complete	1029	12.8	169	266.86 ± 58.91
		[70] w/ Ch. 5	1059870	—	—	—
inst8	8	Complete	1536	2.8	221	367.71 ± 91.65
		[70] w/ Ch. 5	2360832	—	—	—
inst9	9	Complete	2187	0.8	304	375.00 ± 78.67
		[70] w/ Ch. 5	4785156	—	—	—
inst10	10	Complete	3000	0	—	—
		[70] w/ Ch. 5	9003000	—	—	—

* Since the number of obtained solutions is one, the unbiased estimation of standard deviation is undefined.

The parameters in the energy function Eq. (6.11) are set to be $\alpha = 2$, $\beta = 1$, $\delta = 1$, $\gamma = 1$, and $\epsilon = 1$. The simulation parameters are shown in Table 6.3. Since the number of required physical spins for [70] is larger, 10,000 annealing steps for "Complete" is enough, while more steps is needed for " [70]." We tried 500 times for each annealing with different random seeds.

Table 6.2 shows the results of our experiments. In Table 6.2, "Target" column means the target annealing machine and the Ising model embedding method. "Prob. (%)" column means the probability of obtaining the feasible solutions. "Area (best)" and "Area (avg.)" mean the best and average value of WH (area of boundary box) among the feasible solutions along with the unbiased estimation of standard deviation, respectively.

We can see the following as seen in the result. The probability of obtaining the feasible solutions tend to decrease as N increases. As for targeting "Complete," the proposed approach can get the solutions up to $N = 9$. For 20k-spin CMOS annealing

Table 6.3: Simulation parameters.

Parameter	Value
Initial spin value	Random
Spin flipping probability (at the beginning)	0.9
Spin flipping probability (at the end)	0.005
Annealing steps	10,000 (for "Complete") or 100,000 (for " [70]")

Table 6.4: Optimal solutions by the brute-force search.

Name	N	Optimal Area	CPU Time (sec.)
inst1	3	3	0.035
inst2	3	52	0.035
inst3	3	5200	0.038
inst4	4	56	0.072
inst5	5	56	1.290
inst6	6	120	61.007

machine [70] with the embedding method proposed in Chapter 5, the proposed approach can get the solution only $N = 3$. The instance "inst3" contains rectangles which have the rectangles where the width and the height is just 10 times as much as the rectangles in "inst2." Owing to the normalization of the rectangle size, we can get the same floorplan results both in "inst2" and "inst3" and the area of "inst3" is just 100 times as that of "inst2."

6.4.2 Comparison to Brute-Force Search

To check the quality of the solution, we have also implemented a brute-force search algorithm in Python on CentOS 6.8 and Intel Xeon CPU E5-2680 v3 2.50GHz $\times$ 40 machine with 270GB memory. The optimal solutions obtained by our brute-force search algorithm are shown in Table 6.4. Our brute-force search algorithm can obtain the optimal solution up to $N = 6$ ("inst6") within one hour. For $N \geq 7$, our brute-force algorithm cannot obtain the solution within one hour. As seen in Table 6.2 and Table 6.4, our solution by annealing machine successfully reaches the optimal solution up to $N = 5$ ("inst5") for "Complete." For " [70]," the quality of solutions is not good as that of for "Complete."

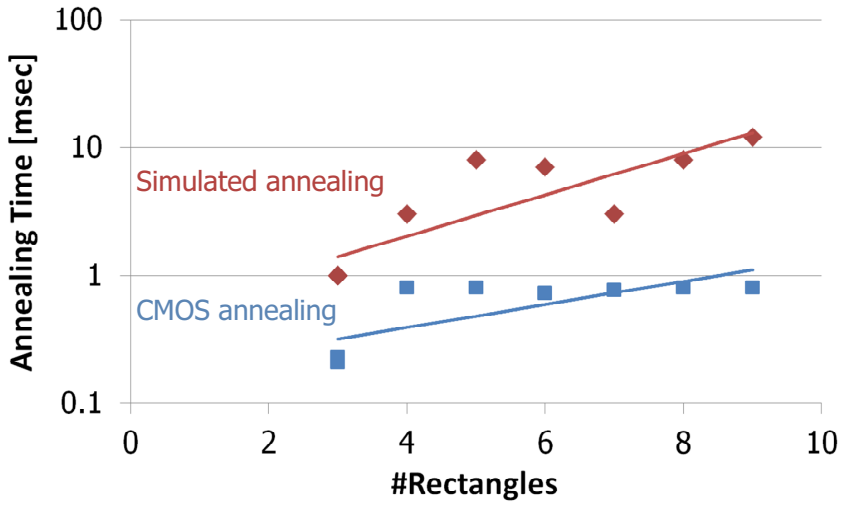


Figure 6.3: Comparison of the required annealing time between simulated annealing and CMOS annealing machine with the same quality of solutions (area).

6.4.3 Comparison to Simulated Annealing

We compare the SA-based approach for solving "Rectangle Packing Problem" described in Section 4.5.4 and CMOS annealing in terms of computational complexity. Firstly, one of the advantages of the SA is that a feasible solution is always obtained by the SA since it is based on the sequence-pair representation. However, a computation time of $O(N^2)$ is needed by using the method proposed in [44] to obtain an area from the sequence-pair, which is a disadvantage of the SA. On the other hand, we only need a computation time of $O(1)$ since the calculation of the cost function is included in the energy function of the Ising model. Secondly, since the SA swaps two selected rectangles in the sequence when performing a transition to an adjacent solution, two rectangles are always changed within one step. However, in the CMOS annealing, the spins corresponding to the all rectangles are updated within one step. Lastly, CMOS annealing machine can be easily parallelized by utilizing some topological characteristics like the 20k-spin CMOS annealing machine, while the SA is difficult to be parallelized. Therefore, CMOS annealing machine has a good scalability against the number of rectangles N compared to the SA-based approach.

We compare computational time of the SA and the CMOS annealing for the "Rectangle Packing Problem" experimentally. For the CMOS annealing machine, we performed annealings with 10000 annealing steps as in Section 6.4.1. For the annealing with the minimum area for each instance, we picked up all spin status for all steps and examined the minimum step having the solution which is the same as that of the final step (required annealing step). We also measured computational time of the SA-based algorithm to get the same quality of solution. In the SA, when an area calculated in each step equals to

or is smaller than the target area, SA is stopped. The other environments of SA is the same as in Section 4.5.4. We compare these computational times. We assume that an Ising model on the CMOS annealing machine can be mapped onto the 20k-spin CMOS annealing machine. When an required annealing step is R on the CMOS annealing machine, the required annealing time is calculated by $R \times 8 \times 10$ [ns]. Figure 6.3 shows the comparison results. As seen in Fig. 6.3, we again confirm that the CMOS annealing machine has a good scalability for the number of rectangles N against the SA-based approach experimentally. We conclude that the solving of "Rectangle Packing Problem" can be accelerated by using the CMOS annealing machines.

6.4.4 Discussion

We can say that an annealing machine with a fully-connected Ising model is reasonably effective for solving "Rectangle Packing Problem." It needs less physical spins and less annealing steps. If it is not practical to implement a fully-connected Ising model in the physical perspective, an annealing machine with more connections between spins is required to solve the real problem. Increasing connections between spins is considered to be an important task. Other tasks such as increasing physical spins and increasing the clock period are the next. For an annealing machine without a fully-connected Ising model, we need an embedding method with shorter length of chains while allowing a small variation in chain length. The method proposed in Chapter 5 generates the same chain length for every logical spins for its simplicity of the algorithm. It is worth considering that an embedding method which ensures the difference between the longest chain length and the shortest one becomes less than K where K is an input parameter, for example.

Even though an annealing machine with fully-connected Ising model is assumed, we can obtain the feasible solution with $N = 9$ at the maximum. This number is considered to be small when solving the real packing problem. The approach in this chapter only recognizes a solution when all the constraints on the oblique grid are satisfied. If we can deal with these unsatisfactions by correcting the spins related to the oblique grid as a post-process of the annealing, we may obtain solutions when N becomes larger. These are ones of the future works.

Improving the mapping to reduce the number of required physical spins is another future work. For example, the connections between the Ising model representing the oblique grid and the Ising model representing the horizontal constraint graph is sparse compared to other connections, since for every i, j, and k, $x_{i,j,k}$ is only connected to $y_{i,j,k}$. The number of connections between $x_{i,j,k}$ and $y_{i,j,k}$ is N^3, which means the connection between this two parts is sparse. Similarly, the connections between the Ising model representing the oblique grid and the Ising model representing the horizontal constraint

graph is sparse. On the other hand, the connections between the Ising model representing the horizontal constraint graph and the vertical one is dense, since for every i, j, and k, $y_{i,j,k}$ is connected to $z_{i',j',k'}$ (for all i', j', k'). The number of connections between $y_{i,j,k}$ and $z_{i,j,k}$ is N^6, which means the connection between this two parts is dense. Creating an embedding method which utilizes this kind of characteristics to reduce the number of required physical spins is another future work.

6.5 Conclusion

In this chapter, we propose an Ising model mapping of "Rectangle Packing Problem" for solving it on the annealing machines. In our proposed mapping, a sequence-pair is represented on an Ising model and the energy function to obtain the optimal solution of the problem is constructed. Our proposed approach maps a "Rectangle Packing Problem" with N rectangles onto a $3N^3$-spin logical Ising model. Experimental results demonstrate that through the proposed approach we can solve the problem with nine rectangles at the maximum on a fully-connected annealing machine and the problem with three rectangles at the maximum on 20k-spin CMOS annealing machine. We can also expect acceleration by annealing machines against the SA-based approach to solve the problem from the experimental results.

Corrections of spins when interpreting them to reduce the unsatisfactions of the constraints as a post-process in the application layer, and improving the mapping to reduce the number of required physical spins are future works. Solving the problem with interconnection wires and/or the problem with allowing rectangle rotations by annealing machines, and comparing the proposed approach to other metaheuristic algorithms (such as SA) are also ones of future works.

Chapter 7

Conclusion

In this dissertation, three floorplan-aware performance-driven HLS algorithms targeting RDR architectures are proposed to cope with the increasing of the interconnection delays. In **Chapter 2**, a floorplan-aware high-level synthesis algorithm with operation chainings based on inter-island distance is proposed. The proposed algorithm enumerates feasible candidates of the operation chaining, and selects the best ones based on maximal chaining distance (MCD). Experimental results demonstrate that the proposed algorithm reduces the latency by up to 40.0% compared to the original one. In **Chapter 3**, a floorplan-aware high-level synthesis algorithm with multiple-operation chainings based on path enumeration is proposed. The proposed algorithm is an extended version of the algorithm proposed in Chapter 2, and enumerates multiple-operation-chaining path candidates. Experimental results demonstrate that the proposed algorithm reduces the latency by up to 30.4% compared to a conventional algorithm, and reduces the latency by up to 24.4% compared to the algorithm proposed in Chapter 2. In **Chapter 4**, a floorplan-driven bitwidth-aware high-level synthesis algorithm using operation chainings is proposed. The proposed algorithm optimizes bitwidths of functional units and utilizes the vacant islands by adding some extra functional units to realize effective operation chainings. Experimental results demonstrate that the proposed algorithm reduces the latency by up to 47% compared to the algorithm in Chapter 2 without area overheads. In summary, these algorithms successfully reduce the latency compared to the conventional algorithms while coping with the increasing interconnection delays. However, the SA-based floorplanning remains the bottleneck in both speed and scalability.

To deal with the bottleneck above, we have tried to apply the floorplanning problem to the forthcoming Ising model-based computers. In **Chapter 5**, a fully-connected Ising model embedding method for 20k-spin CMOS annealing machines is proposed. The proposed method embeds Ising models using less physical spins compared to the de facto standard conventional method in the practical problem size, and that the probability of feasible solutions using the proposed method is better than those of the conventional

method In **Chapter 6**, a mapping of the rectangle packing problem for Ising-model based computers is proposed. Through the proposed mapping we can solve the problem with nine rectangles at the maximum on a fully-connected annealing machine and the problem with three rectangles at the maximum on 20k-spin CMOS annealing machine. We can also expect acceleration by annealing machines against the SA-based approach to solve the problem from the experimental results. We can see a reasonable expectation of using Ising model-based computers for the practical problems, but have some tasks. Corrections of spins when interpreting them to reduce the unsatisfactions of the constraints as a post-process in the application layer, and improving the mapping to reduce the number of required physical spins are ones of the future works. To implement and evaluate our HLS solutions onto FPGA devices is also one of the future works.

Acknowledgment

First and foremost, I would like to give my deepest and most heartfelt thanks to Prof. Nozomu Togawa (戸川望教授) at the Department of Computer Science and Communications Engineering of Waseda University for his all supports on my research work for almost seven years. He has taught me not only research-related techniques and skills but everything about life.

I'm genuinely grateful to Prof. Masao Yanagisawa (柳澤政生教授) at the Department of Electronic and Physical Systems of Waseda University, Prof. Hayato Yamana (山名早人教授) at the Department of Computer Science and Communications Engineering of Waseda University, Prof. Shu Tanaka (田中宗准教授) at Waseda Institute for Advanced Study of Waseda University for their strong support and advice.

I would like to express my gratitude to Prof. Youhua Shi (史又華教授) at the Department of Electronic and Physical Systems of Waseda University for his comments and advice on my research work. I have also received technical support from Dr. Shin-ya Abe (阿部晋矢氏) at Japan Broadcasting Corporation, Dr. Kazushi Kawamura (川村一志助教) at Waseda University, Mr. Yuta Hagio (萩尾勇太氏) at Japan Broadcasting Corporation, and Dr. Masashi Tawada (多和田雅師助教) at Waseda University for my research work.

I also thank Ms. Shuko Watanabe (渡部周子氏) and all the students in the Togawa Laboratory and the Yanagisawa Laboratory for their kindness.

Mr. Masato Hayashi (林真人氏) and Dr. Masanao Yamaoka (山岡雅直氏) at Hitachi, Ltd. have provided us the Ising model simulator. We have had valuable discussions.

Mr. Shusuke Yamada (山田秀祐氏), a former student at the Department of Applied Physics of Tokyo University of Science, and Dr. Yoichiro Hashizume (橋爪洋一郎助教) at the Department of Applied Physics of Tokyo University of Science gently provided us some helpful technical documents.

I would like to offer my special thanks to Mr. Shinichi Takayanagi (高柳慎一氏) at LINE Corporation, Mr. Tomomitsu Motohashi (本橋智光氏) at Sustainable Medicine, and Mr. Kotaro Tanahashi (棚橋耕太郎氏) at Recruit Communications Co., Ltd. for their insightful discussion and great beers.

I have had a wonderful time not to mention in the research life but in leisure time with

my colleagues especially Mr. Keita Igarashi (五十嵐啓太氏) at Hitachi, Ltd., Mr. Manabu Iwasawa (岩澤学氏) at NTT DATA Corporation, Mr. Masaru Oya (大屋優氏) at Waseda University, Ms. Huiqian Jiang (蒋慧倩氏) at NEC Corporation, Mr. Kengo Takeda (竹田健吾氏) at Panasonic Corporation, Mr. Koichi Fujiwara (藤原晃一氏) at NEC Corporation, and Mr. Shinnosuke Yoshida (吉田慎之介氏) at NTT Communications Corporation.

We have encouraged each other and improved ourselves with Mr. Hayate Tanaka (田中颯氏) at Waseda University, Mr. Hirotaka Tamura (田村裕高氏) at Waseda University, and Ms. Senami Furubayashi (古林せなみ氏) at the University of Tokyo.

Mr. Kentaro Nishi (西賢太郎氏) at Yahoo Japan Corporation and Mr. Toshiya Hirohata (廣畑俊哉氏) at Tribox Inc. have told me a lot not only as a great engineer and a great business manager but also as my good friends.

Mr. Norihiro Arita (有田宜弘氏) and Ms. Yuki Matsushita (松下由季氏) at Yahoo Japan Corporation, as well as Mr. Kentaro Nishi, have told me fun and importance of fabrication, which has led me to work as a next job. We had a special time at Hack Day.

We have had a great time with Mr. Yu Nakajima (中島悠氏) at Tribox Inc., Mr. Takafumi Haseda (長谷田貴史氏) at Bank, Inc., Mr. Yuki Kobayashi (古林祐輝氏) at Seiko Watch Corporation, Mr. Shuhei Omura (大村周平氏) at JGC Corporation, and Mr. Kein Takeda (武田慶胤氏) at Tohoku University. I am deeply impressed with their ways of life, and various experiences with them have made me more mature.

Without encouragements and kindness by Mr. Fumiya Matsui (松井郁也氏) at Waseda University, Mr. Ryutaro Miyazaki (宮崎隆太郎氏) at Tokyo Institute of Technology, Ms. Chisaki Fujimaki (藤牧千咲氏) at Tsuda University, and Ms. Eri Tachikawa (立川絵梨氏) at Tsuda University, my research life would not have been so impressive.

My parents and grandparents encourage me all the time.

Finally, this dissertation has been supported partially by JSPS KAKENHI Grand-in-Aid for JSPS Fellows and Waseda Research Institute for Science and Engineering, Grant-in-Aid for Young Scientists (Early Bird).

References

[1] S. Abe, M. Yanagisawa, and N. Togawa, "An energy-efficient high-level synthesis algorithm for huddle-based distributed-register architectures," in *Proc. of 2012 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 576–579, May 2012.

[2] S.-Y. Abe, M. Yanagisawa, and N. Togawa, "Energy-efficient high-level synthesis for HDR architectures," *IPSJ Trans. on System LSI Design Methodology*, vol. 5, pp. 106–117, Aug. 2012.

[3] S.-Y. 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, Sep. 2012.

[4] S.-Y. Abe, "Energy-efficient high-level synthesis algorithms for floorplan-driven SoC architectures," Ph.D. dissertation, Waseda University, Feb. 2015.

[5] AG-Monien, "Graph collection," http://www2.cs.uni-paderborn.de/cs/ag-monien/RESEARCH/PART/graphs.html.

[6] T. Boothby, A. D. King, and A. Roy, "Fast clique minor generation in chimera qubit connectivity graphs," *Quantum Information Processing*, vol. 15, no. 1, pp. 495–508, Jan. 2016.

[7] P. I. Bunyk, E. M. Hoskinson, M. W. Johnson, E. Tolkacheva, F. Altomare, A. J. Berkley, R. Harris, J. P. Hilton, T. Lanting, A. J. Przybysz, and J. Whittaker, "Architectural considerations in the design of a superconducting quantum annealing processor," *IEEE Trans. on Applied Superconductivity*, vol. 24, no. 4, pp. 1–10, Aug. 2014.

[8] J. Cai, B. Macready, and A. Roy, "A practical heuristic for finding graph minors," *arXiv preprint arXiv:1406.2741*, 2014.

[9] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, J. H. Anderson, S. Brown, and T. Czajkowski, "LegUp: High-level synthesis for FPGA-based processor/accelerator systems," in *Proc. of 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, pp. 33–36, 2011.

[10] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, T. Czajkowski, S. D. Brown, and J. H. Anderson, "LegUp: An open-source high-level synthesis tool for FPGA-based processor/accelerator systems," *ACM Trans. on Embedded Compututing Systems*, vol. 13, no. 2, pp. 24:1–24:27, Sep. 2013.

[11] C.-I. Chen and J.-D. Huang, "CriAS: A performance-driven criticality-aware synthesis flow for on-chip multicycle communication architecture," in *Proc. of 2009 Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 67–72, 2009.

[12] C.-I. Chen and J.-D. Huang, "A hierarchical criticality-aware architectural synthesis framework for multicycle communication," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E93-A, no. 7, pp. 1300–1308, Jul. 2010.

[13] L. Chen, M. Ebrahimi, and M. Tahoori, "Reliability-aware operation chaining in high level synthesis," in *Proc. of 2015 20th IEEE European Test Symposium (ETS)*, pp. 1–6, May 2015.

[14] J. Cong, T. Kong, and D. Z. Pan, "Buffer block planning for interconnect-driven floorplanning," in *Proc. of the 1999 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 358–363, 1999.

[15] 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, Apr. 2004.

[16] J. Cong, Y. Fan, and Z. Zhang, "Architecture-level synthesis for automatic interconnect pipelining," in *Proc. of 41st Annual Design Automation Conference (DAC)*, pp. 602–607, 2004.

[17] J. Cong, Y. Fan, G. Han, Y. Lin, J. Xu, Z. Zhang, and X. Cheng, "Bitwidth-aware scheduling and binding in high-level synthesis," in *Proc. of the 2005 Asia and South Pacific Design Automation Conference (ASP-DAC)*, vol. 2, pp. 856–861, Jan. 2005.

[18] 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 Electronic Systems*, vol. 14, no. 3, pp. 35:1–35:31, Jun. 2009.

[19] M. R. Corazao, M. A. Khalaf, L. M. Guerra, M. Potkonjak, and J. M. Rabaey, "Performance optimization using template mapping for datapath-intensive high-level synthesis," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 8, pp. 877–888, Aug. 1996.

[20] P. Coussy, G. Lhairech-Lebreton, and D. Heller, "Multiple word-length high-level synthesis," *EURASIP Journal on Embedded Systems*, vol. 2008, no. 1, p. 916867, 2008.

[21] P. Coussy and A. Morawiec, *High-level synthesis from algorithm to digital circuit*. Springer Netherlands, 2008.

[22] D. Domović and T. Rolich, "Solving strip-packing problem using sequence pair," in *Proc. of 2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pp. 1183–1188, May 2015.

[23] W. C. Elmore, "The transient response of damped linear networks with particular regard to wideband amplifiers," *Journal of Applied Physics*, vol. 19, no. 1, pp. 55–63, Jan. 1948.

[24] D. Eppstein, "Finding large clique minors is hard," *Journal of Graph Algorithms and Applications*, vol. 13, no. 2, pp. 197–204, 2009.

[25] ExPRESS, "Express benchmarks," http://www.ece.ucsb.edu/EXPRESS/benchmark/.

[26] S. Gao, H. Yoshida, K. Seto, S. Komatsu, and M. Fujita, "Interconnect-aware pipeline synthesis for array-based architectures," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E92-A, no. 6, pp. 1464–1475, Jun. 2009.

[27] Y. Hagio, M. Yanagisawa, and N. Togawa, "High-level synthesis with post-silicon delay tuning for RDR architectures," in *Proc. of 2013 International SoC Design Conference (ISOCC)*, pp. 194–197, Nov. 2013.

[28] J.-D. Huang, C.-I. Chen, W.-L. Hsu, Y.-T. Lin, and J.-Y. Jou, "Performance-driven architectural synthesis for distributed register-file microarchitecture with inter-island delay," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E95-A, no. 2, pp. 559–566, Feb. 2012.

[29] Y.-S. Huang, Y.-J. Hong, and J.-D. Huang, "Communication synthesis for interconnect minimization in multicycle communication architecture," *IEICE Trans.*

on Fundamentals of Electronics, Communications and Computer Sciences, vol. E92-A, no. 12, pp. 3143–3150, Dec 2009.

[30] International Technology Roadmap for Semiconductors (ITRS) 2013, http://www.itrs2.net/2013-itrs.html.

[31] M. W. Johnson, M. H. S. Amin, S. Gildert, T. Lanting, F. Hamze, N. Dickson, R. Harris, A. J. Berkley, J. Johansson, P. Bunyk, E. M. Chapple, C. Enderud, J. P. Hilton, K. Karimi, E. Ladizinsky, N. Ladizinsky, T. Oh, I. Perminov, C. Rich, M. C. Thom, E. Tolkacheva, C. J. S. Truncik, S. Uchaikin, J. Wang, B. Wilson, and G. Rose, "Quantum annealing with manufactured spins," *Nature*, vol. 473, no. 7346, pp. 194–198, May 2011.

[32] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[33] K.-I. Kum and W. Sung, "Combined word-length optimization and high-level synthesis of digital signal processing systems," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 8, pp. 921–930, Aug. 2001.

[34] B. Landwehr, P. Marwedel, and R. Dömer, "OSCAR: Optimum simultaneous scheduling, allocation and resource binding based on integer programming," in *Proc. of '94 Conference on European Design Automation (EURO-DAC)*, pp. 90–95, 1994.

[35] B. Le Gal and E. Casseau, "Word-length aware DSP hardware design flow based on high-level synthesis," *Journal of Signal Processing Systems*, vol. 62, no. 3, pp. 341–357, 2011.

[36] S. Lee and K. Choi, "Critical-path-aware high-level synthesis with distributed controller for fast timing closure," *ACM Trans. on Design Automation of Electronics Systems*, vol. 19, no. 2, pp. 16:1–16:29, Mar. 2014.

[37] A. Lucas, "Ising formulations of many NP problems," *Frontiers in Physics*, vol. 2, pp. 1–15, 2014.

[38] T. Ly, D. Knapp, R. Miller, and D. MacMillen, "Scheduling using behavioral templates," in *Proc. of 32nd Conference on Design Automation (DAC)*, pp. 101–106, 1995.

[39] P. Marwedel, B. Landwehr, and R. Dömer, "Built-in chaining: Introducing complex components into architectural synthesis," in *Proc. of '97 Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 599–605, Jan. 1997.

[40] K. Mittal, A. Joshi, and M. Mutyam, "Timing variation-aware scheduling and resource binding in high-level synthesis," *ACM Trans. Design Automation Electronic Systems*, vol. 16, no. 4, pp. 40:1–40:19, Oct. 2011.

[41] T. Miyoshi, http://synthesijer.github.io/web/.

[42] M. C. Molina, J. M. Mendías, and R. Hermida, "Bit-level scheduling of heterogeneous behavioural specifications," in *Proc. of 2002 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 602–608, Nov. 2002.

[43] M. C. Molina, R. Ruiz-Sautua, P. García-Repetto, and J. M. Mendías, "Performance-driven scheduling of behavioural specifications," *VLSI Journal Integration*, vol. 42, no. 3, pp. 294–303, Jun. 2009.

[44] 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, Dec. 1996.

[45] 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. E92-A, no. 12, pp. 3169–3179, Dec. 2009.

[46] T. Okuyama, C. Yoshimura, M. Hayashi, and M. Yamaoka, "Computing architecture to perform approximated simulated annealing for Ising models," in *Proc. of 2016 IEEE International Conference on Rebooting Computing (ICRC)*, pp. 1–8, Oct. 2016.

[47] S. Park and K. Choi, "Performance-driven scheduling with bit-level chaining," in *Proc. of 36th Design Automation Conference (DAC)*, pp. 286–291, 1999.

[48] S. Park and K. Choi, "Performance-driven high-level synthesis with bit-level chaining and clock selection," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 2, pp. 199–212, Feb. 2001.

[49] M. Rim, R. Jain, and R. De Leone, "Optimal allocation and binding in high-level synthesis," in *Proc. of 29th ACM/IEEE Design Automation Conference (DAC)*, pp. 120–123, Jun. 1992.

[50] T. Rolich, D. Domović, and M. Golub, "Bottom-left and sequence pair for solving packing problems," in *Proc. of 2016 39th International Convention on Information*

and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 1318–1323, May 2016.

[51] R. Ruiz-Sautua, M. C. Molina, J. M. Mendías, and R. Hermida, "Behavioural transformation to improve circuit performance in high-level synthesis," in *Proc. of 2005 Design, Automation and Test in Europe (DATE)*, vol. 2, pp. 1252–1257, Mar. 2005.

[52] T. Sadakata and Y. Matsunaga, "A simultaneous module selection, scheduling, and allocation method considering operation chaining with multi-functional units," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E90-A, no. 4, pp. 792–799, Apr. 2007.

[53] Sinby Corporation, http://www.sinby.com/Polyphony/.

[54] S. Sinha and T. Srikanthan, "Dataflow graph partitioning for area-efficient high-level synthesis with systems perspective," *ACM Trans. on Design Automation of Electronic Systems*, vol. 20, no. 1, pp. 5:1–5:18, Nov. 2014.

[55] M. Stephenson, J. Babb, and S. Amarasinghe, "Bitwidth analysis with application to silicon compilation," in *Proc. of the ACM SIGPLAN 2000 Conference on Programming Language Design and Implementation (PLDI)*, pp. 108–120, 2000.

[56] M. Tan, S. Dai, U. Gupta, and Z. Zhang, "Mapping-aware constrained scheduling for LUT-based FPGAs," in *Proc. of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, pp. 190–199, 2015.

[57] 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.

[58] H. Tomiyama, A. Inoue, and H. Yasuura, "Statistical performance-driven module binding in high-level synthesis," in *Proc. of 11th International Symposium on System Synthesis (ISSS)*, pp. 66–71, Dec. 1998.

[59] S. Tsukamoto, M. Takatsu, S. Matsubara, and H. Tamura, "Accelerator architecture for combinatorial optimization problems," *Fujitsu Scientific & Technical Journal*, vol. 53, no. 5, pp. 8–13, Sep. 2017.

[60] University of Toronto, http://legup.eecg.utoronto.ca/.

[61] D. Venturelli, S. Mandrà, S. Knysh, B. O'Gorman, R. Biswas, and V. Smelyanskiy, "Quantum optimization of fully connected spin glasses," *Physical Review X*, vol. 5, no. 3, pp. 031 040:1–031 040:8, Sep. 2015.

[62] D. Venturelli, D. Marchand, and G. Rojo, "Job shop scheduling solver based on quantum annealing," in *Proc. of ICAPS-16 Workshop on Constraint Satisfaction Techniques for Planning and Scheduling (COPLAS)*, pp. 25–34, Jun. 2016.

[63] Y.-H. Wu, C.-J. Yu, and S.-D. Wang, "Heuristic algorithm for the resource constrained scheduling problem during high-level synthesis," *IET Computers Digital Techniques*, vol. 3, no. 1, pp. 43–51, Jan. 2009.

[64] Xilinx, Inc., http://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html.

[65] Xilinx, Inc., https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html.

[66] Xilinx, Inc., http://www.pynq.io/.

[67] S. Xydis, I. Triantafyllou, G. Economakos, and K. Pekmestzi, "Flexible datapath synthesis through arithmetically optimized operation chaining," in *Proc. of 2009 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pp. 407–414, Jul. 2009.

[68] S. Yamada, Y. Hashizume, T. Nakajima, and S. Okamura, "Solving a rectangular packing problem using quantum annealing (in Japanese)," in *Proc. IEICE-ISS Poster Session at IEICE General Conference 2017*, p. 30, Mar. 2017.

[69] M. Yamaoka, C. Yoshimura, M. Hayashi, T. Okuyama, H. Aoki, and H. Mizuno, "20k-spin Ising chip for combinational optimization problem with CMOS annealing," in *2015 IEEE International Solid-State Circuits Conference (ISSCC) Digest of Technical Papers*, pp. 1–3, Feb. 2015.

[70] M. Yamaoka, C. Yoshimura, M. Hayashi, T. Okuyama, H. Aoki, and H. Mizuno, "A 20k-spin Ising chip to solve combinatorial optimization problems with CMOS annealing," *IEEE Journal of Solid-State Circuits*, vol. 51, no. 1, pp. 303–309, Jan. 2016.

[71] Y. Yi, I. Nousias, M. Milward, S. Khawam, T. Arslan, and I. Lindsay, "System-level scheduling on instruction cell based reconfigurable systems," in *Proc. of 2006 Design, Automation and Test in Europe (DATE)*, vol. 1, pp. 1–6, Mar. 2006.

[72] C. Yoshimura, M. Yamaoka, M. Hayashi, T. Okuyama, H. Aoki, K. Kawarabayashi, and H. Mizuno, "Uncertain behaviours of integrated circuits improve computational performance," *Scientific Reports*, vol. 5, Nov. 2015.

[73] C. Yoshimura, M. Hayashi, T. Okuyama, and M. Yamaoka, "FPGA-based annealing processor for Ising model," in *Proc. of 2016 Fourth International Symposium on Computing and Networking (CANDAR)*, pp. 436–442, Nov. 2016.

[74] D. C. Zaretsky, G. Mittal, R. Dick, and P. Banerjee, "Balanced scheduling and operation chaining in high-level synthesis for FPGA designs," in *Proc. of 8th International Symposium on Quality Electronic Design (ISQED)*, pp. 595–601, Mar. 2007.

[75] R. Zhao, M. Tan, S. Dai, and Z. Zhang, "Area-efficient pipelining for FPGA-targeted high-level synthesis," in *Proc. of 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, Jun. 2015.

List of Publications

論文（学術誌原著論文）

⟨1⟩ ○ **K. Terada**, M. Yanagisawa, and N. Togawa, "A bitwidth-aware high-level synthesis algorithm using operation chainings for Tiled-DR architectures," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E100-A, no. 12, pp. 2911–2924, Dec. 2017.

⟨2⟩ ○ **K. Terada**, M. Yanagisawa, and N. Togawa, "A high-level synthesis algorithm with inter-island distance based operation chainings for RDR architectures," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E98-A, no. 7, pp. 1366–1375, Jul. 2015.

国際会議

⟨3⟩ ○ **K. Terada**, M. Yanagisawa, and N. Togawa, "A floorplan-driven high-level synthesis algorithm with multiple-operation chainings based on path enumeration," in *Proceedings of 2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2129–2132, Lisbon, Portugal, May 2015.

⟨4⟩ ○ **K. Terada**, M. Yanagisawa, and N. Togawa, "A floorplan-driven high-level synthesis algorithm with operation chainings using chaining enumeration," in *Proceedings of 2014 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, pp. 248–251, Ishigaki, Japan, Nov. 2014.

国内学会

⟨5⟩ 寺田晃太朗, 田中宗, 林真人, 山岡雅直, 柳澤政生, 戸川望, "20K スピン CMOS アニーリングマシンを対象とした完全結合イジングモデルマッピング手法," 日本物理学会 2017 年秋季大会, 盛岡市, Sep. 2017.

⟨6⟩ (査読あり) 寺田晃太朗, 田中宗, 林真人, 山岡雅直, 柳澤政生, 戸川望, "20K スピン CMOS アニーリングマシンを対象とした完全結合イジングモデルマッピ

ング手法と評価," 情報処理学会 DA シンポジウム 2017 論文集, pp. 163–168, 加賀市, Sep. 2017.

⟨7⟩ (ポスター発表) 長谷川健人, 石川遼太, 寺田晃太朗, 川村一志, 多和田雅師, 戸川望, "組込みデバイスと FPGA を用いたナンバーリンクソルバの設計と実装," 情報処理学会 DA シンポジウム 2017 ポスター発表, 加賀市, Aug. 2017.

⟨8⟩ 寺田晃太朗, 柳澤政生, 戸川望, "演算ビット幅に基づく演算チェイニングを用いた RDR アーキテクチャ向け性能指向高位合成手法," 電子情報通信学会 2016 年ソサイエティ大会講演論文集, p. 71, 札幌市, Sep. 2016.

⟨9⟩ (ポスター発表) 寺田晃太朗, 長谷川健人, 川村一志, 多和田雅師, 戸川望, "機械学習と FPGA を用いたナンバーリンクソルバ," 情報処理学会 DA シンポジウム 2016 ポスター発表, 加賀市, Sep. 2016.

⟨10⟩ 寺田晃太朗, 柳澤政生, 戸川望, "DFG のクリティカルパス最適化に基づく演算チェイニングを用いた RDR アーキテクチャ対象高位合成手法," 信学技報, VLD2016-05, pp. 41–46, 北九州市, May 2016.

⟨11⟩ (ポスター発表) 寺田晃太朗, 川村一志, 多和田雅師, 藤原晃一, 戸川望, "機械学習を用いたナンバーリンクソルバ," 情報処理学会 DA シンポジウム 2015 ポスター発表, 加賀市, Aug. 2015.

⟨12⟩ (査読あり) 寺田晃太朗, 柳澤政生, 戸川望, "演算チェイニングの候補列挙・選択アルゴリズムを用いたフロアプラン指向高位合成手法," 情報処理学会 DA シンポジウム 2015 論文集, pp. 17–22, 加賀市, Aug. 2015.

⟨13⟩ (査読あり) 寺田晃太朗, 柳澤政生, 戸川望, "多段演算チェイニングを利用した配線遅延を考慮した高位合成手法," 情報処理学会 DA シンポジウム 2014 論文集, pp. 115–120, 下呂市, Aug. 2014.

⟨14⟩ (査読あり) 寺田晃太朗, 柳澤政生, 戸川望, "演算チェイニング候補列挙に基づく配線遅延を考慮した高位合成手法," 第 27 回回路とシステムワークショップ論文集, pp. 440–445, 淡路市, Aug. 2014.

業績賞等

⟨15⟩ 2017 年 8 月 情報処理学会 DA シンポジウム 2017 アルゴリズムデザインコンテスト 最優秀賞.

⟨16⟩ 2016 年 9 月 情報処理学会 DA シンポジウム 2016 アルゴリズムデザインコンテスト 最優秀賞.

⟨17⟩ 2016 年 9 月 情報処理学会 DA シンポジウム 2016 アルゴリズムデザインコンテスト 特別賞.

⟨18⟩ 2016 年 9 月 情報処理学会 第 176 回システムと LSI の設計技術研究発表会 優秀発表学生賞.

⟨19⟩ 2015 年 8 月 情報処理学会 DA シンポジウム 2015 アルゴリズムデザインコンテスト 優秀賞 (学生部門).

研究費・助成金

⟨20⟩ 早稲田大学理工学術院総合研究所, 若手研究者支援事業アーリーバード, “超低消費電力を実現する革新的アニーリングマシンアプリケーション技術,” 2017 年 6 月–2018 年 3 月, 総額 80 万円.

⟨21⟩ 日本学術振興会特別研究員奨励費, “高性能かつ小面積を実現するフロアプランを考慮した LSI 上位設計技術,” 2016 年 4 月–2018 年 3 月, 総額 190 万円 (2016 年度: 100 万円, 2017 年度:90 万円).