IPSJ Transactions on System LSI Design Methodology Vol. 2 43–52 (Feb. 2009)

Regular Paper

Binding Refinement for Multiplexer Reduction

Sho Kodama[†1] and Yusuke Matsunaga[†2]

In behavioral synthesis for resource shared architecture, multiplexers are inserted between registers and functional units as a result of binding if necessary. Multiplexer optimization in binding is important for performance, area and power of a synthesized circuit. In this paper, we propose a binding algorithm to reduce total amount of multiplexer ports. Unlike most of the previous works in which binding is performed by a constructive algorithm, our approach is based on an iterative improvement algorithm. Starting point of our approach is initial functional unit binding and initial register binding. Both functional unit binding and register binding are modified by local improvements based on taboo search iteratively. The binding in each iteration is feasible, hence actual value of total amount of multiplexer ports can be optimized. The smart neighborhood which considers an effect of sharing of connection is used in the proposed method for effective reduction of total amount of multiplexer ports. Additionally, the massive modification of binding is performed by regular intervals to achieve a further reduction of total amount of multiplexer ports and further robustness for an initial binding. Experimental results show that our approach can reduce total amount of multiplexer ports by 30% on an average compared to a traditional binding algorithm with computation time of several seconds to a few minutes. Also, results of robustness evaluation show that our approach barely depends on initial binding.

1. Introduction

In recent years, cost and period for designing VLSI (Very Large Scale Integrated Circuit) have risen due to advancement of integration. One of the solutions for this problem is to use behavioral synthesis which generates RTL (Register Transfer Level) description from abstract description called behavioral description, automatically.

Meanwhile, in behavioral synthesis for resource shared architecture, multiplexers (MUX) are inserted between registers and functional units (FU) as a result of binding if necessary. MUX should be optimized because of its significant effect on area, performance, and power of a synthesized circuit.

Generally, four tasks are performed sequentially in behavioral synthesis: module selection, module allocation, scheduling, and binding. Module selection and module allocation determine types and the number of FUs which is used for each operation; Scheduling determines the clock cycle at which each operation will be executed; Binding assigns each operation and each variable to the instance of FU and the register, respectively. Generally, FU binding and register binding are performed separately in binding. FU binding assigns each operation to a particular instance of FU. Register binding assigns each variable to a particular register. Although each task of behavioral synthesis affects generation of MUX, this paper assumes that module selection, module allocation and scheduling are already finished. Focus of this paper is binding because it significantly affects MUX optimization. The problem to find a binding which minimizes MUX is proved to be NP-hard even if either FU binding or register binding is finished and fixed [1].

Due to its computational complexity, practical solutions of this problem are based on some heuristics. In earlier works, FU binding and register binding based on clique partitioning or branch-and-bound search were proposed [2)–4)]. The minimum number of registers is not guaranteed in these methods. Later on, the FU binding and the register binding based on weighted bipartite matching (LYRA and ARYL) were proposed [5]. The weight which corresponds to the probability of generating MUX is assigned to each edge of bipartite graph. This approach guarantees the minimum number of registers. The binding algorithm using minimal cost network flow algorithm instead of weighted bipartite matching was also proposed [6]. Register binding under the condition in which FU binding is assumed to be finished was also solved by k-cofamily algorithm [7]. Among the above-mentioned previous works, FU binding and register binding are performed sequentially, for example, register binding follows FU binding. MUXs are generated from correlation between FU binding and register binding, hence it is difficult to optimize MUX globally with sequentially performed FU binding and register binding. Kim and Liu proposed simultaneous scheduling, FU binding and register binding by every clock cycle fashion to solve this issue [8]. Experimental

†1 Graduate School of Information Science and Electrical Engineering, Kyushu University
†2 Faculty of Information Science and Electrical Engineering, Kyushu University

results show that the total amount of MUX ports is less than that of LYRA and ARYL [5], however, the number of registers is increased. This method can consider correlation between FU binding and register binding better than previous methods, but optimization of MUX remains to be limited to operations and variables bound in each clock cycle. Simultaneous FU binding and register binding (SFR) which guarantee the minimum number of registers have been recently proposed [9]. SFR starts from the binding in which each FU and each register are unshared, namely, the number of FUs and the number of registers equal to the number of operations and the number of registers, respectively. Interleaved FU binding and register binding are then performed iteratively with decreasing the number of FUs and registers until resource constraint. FU binding and register binding can optimize MUX globally with the probability of generating MUX and information from previously performed FU binding and register binding. This method accomplishes better results than previous method [6] in total area of MUX and clock period. The concern is that SFR is unable to apply the behavioral description which includes a control such as conditional branching.

The above-mentioned previous algorithms other than SFR [9] start FU binding and register binding from the state in which none of operation and variable is assigned to a FU and a register, respectively. The k-cofamily based algorithm [7] starts from the situation in which FU binding is finished, but the register binding is performed from the same state as the above. Feasible solutions of FU binding and register binding are not achieved until these algorithms terminate. As a result, the number of input ports, the area, and the delay of each MUX generated are not determined until they finally construct a feasible solution. Previous methods, therefore, have no alternative but to use local information which are likely connected to MUX optimization. SFR can use information from the FU binding and the register binding constructed in previous iteration, but the solution achieved in each iteration is not a feasible solution because the number of FUs and the number of registers violate resource constraint. There is no guarantee that the information for optimizing MUX produced by the previously constructed FU binding and register binding provide desirable effect to final feasible solution. Accordingly, MUX optimization remains to perform with local information even in SFR.

One of alternative approaches to optimize MUX in binding is using an iterative improvement algorithm. It starts from a feasible FU binding and a feasible register binding. Local improvements are then applied to them iteratively to optimize MUX. Specifically, a FU and a register to which each operation and each variable are assigned, respectively are modified. This approach can optimize MUX directly because the FU binding and the register binding in each iteration is feasible, and MUXs generated are accurately observed. Few binding algorithms based on an iterative improvement algorithm were proposed. Some binding algorithms based on simulated annealing were proposed [10),11]. The concern is their long computation time. One of the most critical issues for using an iterative improvement algorithm is a method of creating neighborhoods. A neighborhood is the binding created with modification of the current binding. Generally, an iterative improvement algorithm generates multiple neighborhoods in each iteration except for probabilistic meta algorithms such as simulated annealing and genetic algorithm. The neighborhood which has the best gain to an objective function in all neighborhoods generated, is replaced with the current binding. Consequently, the method of creating neighborhoods which leads to high quality of MUX optimization, high robustness for initial binding, and short computation time, is required.

In this paper, the binding method based on an iterative improvement algorithm to minimize total amount of MUX ports (MUX Cost) is proposed. Both FU binding and register binding are modified by local improvements based on taboo search iteratively. The smart neighborhood, named connection driven-neighborhood which considers an effect of sharing of connection is used in the proposed method for effective reduction of MUX Cost. This neighborhood is based on the fact that elimination of the connection for reducing MUX Cost can be accomplished with generating the modified binding in which none of variable uses the connection. The set of operations which consume, or generate variables which share the same connection is moved or swapped to generate connection driven-neighborhood. The set of variables which share the same connection is also adopted to generate connection driven-neighborhood. The proposed method can achieve significant reduction of MUX Cost, short computation time, and high robustness for an initial binding with this neighborhood. Additionally, the mas-

sive modification of FU binding and register binding based on weighted bipartite matching is performed at regular intervals. The purposes of this procedure are to achieve a further reduction of MUX Cost and further robustness for an initial binding. Experimental results show that our approach is able to reduce MUX Cost by about 30% on an average compared to a traditional binding algorithm with computation time of several seconds to a few minutes. Also, results of robustness evaluation in which our approach is applied to randomly generated binding show that our approach barely depends on initial binding.

Structure of this paper is as follows: In Section 2, preliminaries and problem definitions are introduced; In Section 3, proposed binding method is presented; In Section 4, we show some experimental results, and Section 5 concludes this paper.

2. Preliminaries and Definitions

In behavioral synthesis, behavioral description is transformed to CDFG (Control Data Flow Graph) which consists of single CFG (Control Flow Graph) and one or more DFG (Data Flow Graph). DFG is directed acyclic graph which represents data flow of a circuit. Each node of DFG represents operation, and each edge between operations represents data dependency. CFG is directed graph which represents control flow of a circuit. Each node of CFG corresponds to one DFG. An input of binding is scheduled DFG. The set of all operations to be assigned to FU is denoted as O, and the set of all variables generated by operations is denoted as V. The set of registers is denoted as R, and the set of FU is denoted as F.

Two operations $o_i, o_j \in O$ can be assigned to the same FU only if their executed clock cycles are not overlapped. The lifetime of variable v is the interval of clock cycle between v is generated and v is consumed. Two variables $v_i, v_j \in V$ can be assigned to the same register only if their lifetimes are not overlapped. This paper assumes that each DFG is constructed such that operations which belong to different DFGs are not executed concurrently. Two operations which belong to different DFGs, therefore, can be assigned to the same FU. Additionally, two variables which belong to different DFGs can be assigned to the same register. If two operations o_i, o_j can be assigned to the same FU, this binary relation is denoted as $o_i \approx o_j$. If two variables v_i, v_j can be assigned to the same register, this binary relation is denoted as $v_i \approx v_j$. Consequently, FU binding $g_o : O \to F$ and register binding $g_v : V \to R$ must satisfy equations as follows:

$$\forall f_i \in F\ ,\ \forall o_j, o_k \in O_b(f_i)\ ,\ j \neq k\ ,\ o_j \approx o_k \tag{1}$$

$$\forall r_i \in R\ ,\ \forall v_j, v_k \in V_b(r_i)\ ,\ j \neq k\ ,\ v_j \approx v_k \tag{2}$$

Where $O_b(f_i)$ is the set of operations assigned to FU f_i, and $V_b(r_i)$ is the set of variables assigned to register r_i. These sets can be defined as follows:

$$O_b(f_i) = \{o | o \in O\ ,\ g_o(o) = f_i\} \tag{3}$$

$$V_b(r_i) = \{v | v \in V\ ,\ g_v(v) = r_i\} \tag{4}$$

The MUX Cost at FU binding g_o and register binding g_v is denoted as $mux(g_o, g_v)$. MUX Cost $mux(g_o, g_v)$ is defined as follows:

$$mux(g_o, g_v) = \sum_{r_i \in R} \Big| \bigcup_{v_j \in V_b(r_i)} p_{source}(v_j) \Big| + \sum_{f_i \in F} \sum_{p_j \in P_{in}(f_i)} \Big| \bigcup_{v_k \in V_{cons}(p_j)} g_v(v_k) \Big| \tag{5}$$

Where $p_{source}(v_i)$ represents the output port of FU which generates variable v_i, and $P_{in}(f_i)$ represents the set of input ports of FU f_i. $V_{cons}(p_i)$ represents the set of variables which is consumed by port p_i.

To calculate $p_{source}(v_i)$ and $V_{cons}(p_i)$ in Eq. (5), port assignment which determines correspondence relation between each variable and input or output port of FU must be performed. Port assignment determines the input port of FU which consumes each variable, and the output port of FU which generates each variable. Port assignment affects MUX Cost, however, this paper assumes that it is already finished and fixed in terms of computation time. An alteration of port assignment in binding is our future work. Some refinement algorithms which modify port assignment to improve MUX Cost can be applied after FU binding and register binding are finally determined [7),12)].

This paper assumes that the set of FU F and the set of registers R are already determined and fixed. Hence, problem definition in this paper is as follows:

- Input: O, V, F, and R
- Objective: Find FU binding g'_o and register binding g'_v which minimize $mux(g'_o, g'_v)$ with satisfying Eq. (1) and Eq. (2)

3. Binding Refinement

A local improvement based on taboo search is applied to given initial FU binding and initial register binding iteratively for N times in our approach. The flow of our approach is shown in **Fig. 1**. The current FU binding and the current register binding are alternately modified in the proposed method. Modifications of the current FU binding and the current register binding are performed as follows: first, multiple neighborhoods are generated from the current binding; the neighborhood which has the best gain to MUX Cost is then selected from them; finally, the current binding is replaced by the selected neighborhood. A neighborhood is the FU binding g'_o and the register binding g'_v generated with modification of the current binding (g_o, g_v). Additionally, the massive modification for binding based on weighted bipartite matching is applied every $loop_{wb}$ iterations. After all iterations are finished, the best binding whose MUX Cost is minimal in all iterations is output. In this section, each procedure of the proposed method is introduced.

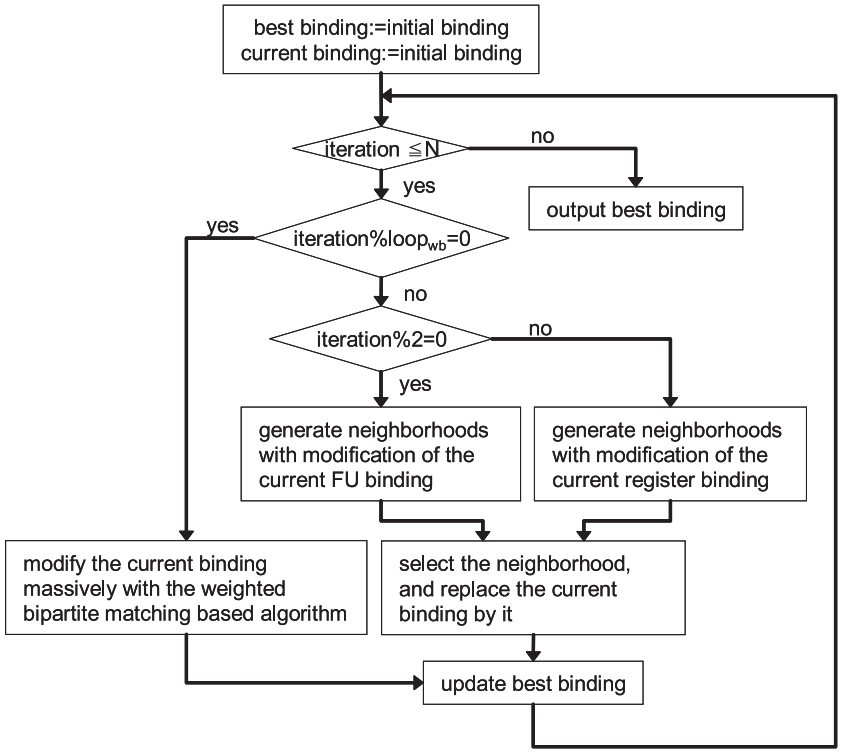

Fig. 1 Flow of improvement.

3.1 Neighborhood Enumeration

Neighborhoods enumerated in each iteration are generated with modification of the current FU binding g_o or the current register binding g_v, namely, either the current FU binding or the current register binding is modified in each iteration.

To make search process for optimizing MUX efficient, the neighborhood which considers an effect of sharing of connection between FU and register is adopted in the proposed method. This neighborhooed is named connection driven-neighborhood (cd-neighborhood). A cd-neighborhood is based on the fact that elimination of the connection for reducing MUX Cost can be accomplished with generating the modified binding in which none of a variable uses the connection. This means that to eliminate the connection shared by multiple variables with modification of FU binding, operations which consume or generate these variables need to be simultaneously moved to another FU. Similarly, to eliminate the connection shared by multiple variables with modification of register binding, these variables need to be simultaneously moved to another register. Otherwise, the connection shared by multiple variables cannot be eliminated in single iteration of an iterative improvement algorithm.

We will introduce it by using the binding example shown in **Fig. 2** (a). In this example, p_1, p_2 represent input ports of each operation, and $r_i (i \in [1,5])$ represent registers. $o_i (i \in [1,3])$ represent operations, and f_1 represent FU. p'_1, p'_2 represent input ports of FU f_1. Variables consumed by input ports p_1, p_2 of each operation are assumed to be consumed by input ports p'_1, p'_2 of FU f_1, respectively. Two input MUXs mux_1, mux_2 are generated in the data path structure of Fig. 2 (a), as shown in Fig. 2 (b). Let us discuss the situation in which MUX mux_1 need to be eliminated with modification of FU binding. In this situation, connection $r_2 \rightarrow p'_2$ or $r_3 \rightarrow p'_2$ must be eliminated. To eliminate connection $r_3 \rightarrow p'_2$, operation o_3 just must be moved to other FU. To eliminate

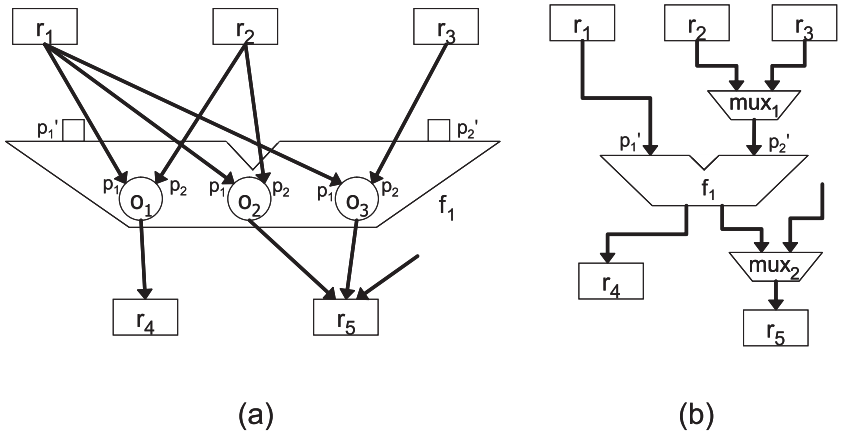

Fig. 2 (a) A binding example for neighborhood enumeration, (b) data path of a binding example.

connection $r_2 \rightarrow p'_2$, however, operation o_1 and o_2 must be simultaneously moved to other FU because this connection is shared by variables consumed by input port p'_2 of FU f_1. Similar situation is occurred when connection $f_1 \rightarrow r_5$ need to be eliminated.

Consequently, in a modification of the current FU binding, the set of operations $O_{TOS} \subseteq O_b(f_i)$, called Targeted Operation Set (TOS) which satisfies Eq. (6) or Eq. (7) is enumerated for each FU f_i. The FU to which each operation in a TOS assigned, is modified in generation of cd-neighborhoods.

$$\sum_{p_j \in P_{in}(f_i)} \left| \bigcap_{o_k \in O_{TOS}} g_v(V_{expend}(o_k) \cap V_{cons}(p_j)) \right| = numin(f_i) \tag{6}$$

$$\forall o_j, o_k \in O_{TOS}\ ,\ g_v(v_{gen}(o_j)) = g_v(v_{gen}(o_k)) \tag{7}$$

Where $numin(f_i)$ represents the number of input ports of FU f_i. $V_{expend}(o_i)$ represents the set of variables which is consumed by operation o_i, and $v_{gen}(o_i)$ represents the variable which is generated by operation o_i. Each variable $v_i \in V_{expend}(o_j)$ is consumed by different input port of FU each other, hence element count of the set $V_{expend}(o_j) \cap V_{cons}(p_k)$ in Eq. (6) equals to one. Note that the connection used by single variable can be eliminated with the cd-neighborhood for TOS. Equations (6) and (7) also consider an unshared connection.

All TOSs enumerated for each FU f_i are held in the list $list_{TOS}(f_i)$ which forbids duplication. In the case of Fig. 2 (a), the list of TOS $list_{TOS}(f_1)$ is defined as follows:

$$list_{TOS}(f_1) = \{\{o_1, o_2\}, \{o_3\}, \{o_2, o_3\}, \{o_1\}\} \tag{8}$$

After the list of TOS of each FU f_i is generated, all TOSs in each list are sorted in ascending order according to their element count. Next, all elements are deleted, except k elements from beginning of $list_{TOS}(f_i)$ to reduce the number of cd-neighborhoods generated. k is integer number defined as $k = round(ratio \times list_{TOS}(f_i).size)$, however, it is adjusted as $k \geq 1$. Parameter $ratio$ is positive number, and it is decreased by $delta_{ratio}$ only if the best binding is updated. Also, it is increased by $delta_{ratio}$ only if the best binding is not updated for $loop_{ratio}$ iterations. However, $ratio$ is adjusted as $min_{ratio} \leq ratio \leq 1.00$. Initial value of $ratio$ is 1.00.

Finally, cd-neighborhoods for each TOS in $list_{TOS}(f_i)$ are generated in which each TOS is moved to other FU $f_j (i \neq j)$ or two TOSs bound to different FUs are swapped each other, if possible. At generating of each cd-neighborhood, the gain to MUX Cost denoted as $gain(g'_o, g'_v)$ is calculated as follows:

$$gain(g'_o, g'_v) = mux(g_o, g_v) - mux(g'_o, g'_v) \tag{9}$$

Where g'_o, g'_v are FU binding and register binding of the cd-neighborhood. Each cd-neighborhood and its gain are held to the list.

In a modification of the current register binding, the set of variables $V_{TVS} \subseteq V_b(r_i)$ called TVS (Targeted Variable Set), is enumerated by each register $r_i \in R$, and then cd-neighborhood for each TVS is generated. TVS $V_{TVS} \subseteq V_b(r_i)$ satisfies Eq. (10) or Eq. (11).

$$\forall v_j, v_k \in V_{TVS}\ ,\ p_{source}(v_j) = p_{source}(v_k) \tag{10}$$

$$\forall v_j, v_k \in V_{TVS}\ ,\ P_{sink}(v_j) \cap P_{sink}(v_k) \neq \phi \tag{11}$$

Where $P_{sink}(v_i)$ is the set of input ports of FU which consume variable v_i. A TVS is the set of variables which share the same connection. The following of process is the same as the case of modification for the current FU binding.

3.2 Massive Modification of the Current Binding

To avoid stagnation of search process, the massive modification of the current binding based on weighted bipartite matching is performed by every $loop_{wb}$ iter-

ations. The input binding of this modification is determined as the best binding only if it was updated after this modification was previously performed. Otherwise, the input binding is determined as the current binding. The input binding is iteratively modified with following process for $iteration_{wb}$ times: first, register binding is reset and all variables are assigned to register again by constructive algorithm, called Register Binding Bipartite (RBB); FU binding is then reset and all operations are assigned to FU again by constructive algorithm, called FU Binding Bipartite (FBB). After that, the binding whose MUX Cost is minimal in all binding generated by RBB or FBB of each iteration is recorded. Note that, the input binding is never recorded.

RBB and FBB are based on weighted bipartite matching also used in some previous works [5),13)]. RBB performs following process sequentially. First, all variables are sorted in ascending order with their start clock cycle of lifetime, as the primary key, and in descending order with their end clock cycle of lifetime, as the secondary key. Sorted variables are then divided to the set of variables, which cannot be assigned to the same register, called cluster, and then binding of variables is performed by each cluster. By each cluster, weighted bipartite graph $G = (V_{wb} \cup R, E)$ is constructed. V_{wb} is the set of variables included in cluster. If variable $v_i \in V_{wb}$ can be assigned to register $r_j \in R$, edge $e_{ij} \in E$ exists. The weight w_{ij} assigned to the edge e_{ij} is defined as increase of the MUX Cost when the variable v_i is assigned to the register r_j. Next, minimal cost maximal matching M of the weighted bipartite graph G is solved by Hungarian Method [14)] with $O(|R|^3)$ computational complexity. For each edge $e_{mn} \in M$, variable v_m is assigned to register r_n. Process of FBB is the same as RBB.

The resultant binding depends on the input binding in this procedure because FU binding is not reset in RBB and register binding is not reset in FBB. Also, this procedure is constructed for optimizing MUX Cost, hence a massive modification of FU binding and register binding can be achieved without great increase of MUX Cost.

3.3 Neighborhood Selection

The current binding is replaced by the cd-neighborhood selected from enumerated cd-neighborhoods, called Accepted Neighborhood in each iteration. Accepted Neighborhood is determined as follows: the cd-neighborhood, which has the best gain, and is not forbidden by taboo-list. If there are multiple cd-neighborhoods which have the same gain, following value is used for the second measure: average of times each operation or each variable which is moved or swapped to generate these cd-neighborhoods, was used to generate previous Accepted Neighborhoods. Also, if the best binding will be updated by selection of the cd-neighborhood $neib_i$, $neib_i$ is certainly selected even if it is forbidden by taboo-list. After Accepted Neighborhood is selected, the current binding is replaced by it.

Taboo-list is a fixed size queue used to avoid cycling of search process. A forbidden modification of binding, called taboo is pushed to taboo-list in each iteration. The taboo is defined as follows in the proposed method: operations or variables, which are moved or swapped for generating Accepted Neighborhood are assigned to the FU or the register to which they are assigned in the current binding. For example, if the set of variables $\{v_1, v_2\}$ are moved from register r_3 to other register to generate Accepted Neighborhood, the taboo is that variable v_1 or v_2 is assigned to register r_3. Also, taboo-list for operations and that for variables are used because modification of FU binding and that of register binding are interleaved. Sizes of these taboo-lists are the same.

4. Experimental Results

In Section 4.1, the results of a comparison between the proposed method and previous work are presented. In Section 4.2, robustness for initial binding of the proposed method is evaluated.

4.1 Comparison to Previous Work

To evaluate the efficiency of our approach, comparison between the proposed method and LYRA [5)] were performed with some benchmarks shown in **Table 1**. Although, comparison between the proposed method and SFR [9)] is also desirable, we could not do the fair comparison due to the following reasons. First, it was difficult to handle experiments under the same condition as SFR. Second, the cost function used in binding was not specified in SFR.

Two parameters mainly affect the MUX Cost of finally achieved binding in the proposed method. One is the number of iterations denoted as N, and the other is $loop_{wb}$ which determines repetition interval of the massive modification of the

Table 1 Benchmark.

	$\|V\|$	$\|O\|$	$\|R\|$
jpeg_dct	116	98	21
blowfish_encrypt	325	243	19
sobel_filter	56	45	10
jacobi	304	270	73
diff_eq	24	11	6

Table 2 Parameters of the proposed method.

Size of taboo-list	10
min_{ratio}	0.3
$delta_{ratio}$	0.05
$loop_{ratio}$	100
$iteration_{wb}$	2

current binding. Value of N directly affects MUX Cost and computation time. Large value of N generally achieves significant reduction of MUX Cost, but a great deal of computation time will be required. Value of N must be determined by considering trade-off between reduction of MUX Cost and computation time. Value of $loop_{wb}$ also affects MUX Cost and computation time because repetition of the massive modification of the current binding is determined by it. For these reasons, we evaluated multiple combinations of different values of N and $loop_{wb}$.

Other parameters of the proposed method are shown in **Table 2**. The initial binding of the proposed method was generated with LYRA. The number of each type of FU was defined as $round(0.7 \times Num_{ASAP})$, where Num_{ASAP} is the number of FU derived from ASAP (As Soon As Possible) scheduling. Each operation in CDFG was scheduled by list scheduling. The proposed method and LYRA were implemented in C++ language, and evaluated in Xeon 5140 (2.33 GHz Dual Core), 8 GB main memory server.

Experimental results of MUX Cost and computation time under $loop_{wb} = 1{,}000$ are shown in **Table 3**. Value of sixth column is the normalized MUX Cost of the proposed method based on LYRA. The proposed method reduces MUX Cost by 28% on an average and 40% tops compared to LYRA. Also, computation time of the proposed method is several seconds to a few minutes, and it is well within utility. Table 3 shows trade-off between reduction of MUX Cost and computation time. The MUX Cost and the computation time of the proposed method evaluated in multiple combinations of different values of N and $loop_{wb}$

Table 3 MUX Cost and computation time under $loop_{wb} = 1{,}000$.

Benchmark	LYRA		LYRA+Proposed			
	MUX Cost	run time [s]	N	MUX Cost	Norm.	run time [s]
jpeg_dct	132	0.1	2,500	92	0.70	3.3
			5,000	92	0.70	12.6
			7,500	91	0.69	22.0
			10,000	91	0.69	31.3
blowfish_encrypt	112	0.5	2,500	88	0.79	6.0
			5,000	88	0.79	13.0
			7,500	88	0.79	21.0
			10,000	88	0.79	28.5
sobel_filter	61	0.0	2,500	48	0.79	1.2
			5,000	47	0.77	2.4
			7,500	47	0.77	3.8
			10,000	47	0.77	5.2
jacobi	421	4.1	2,500	272	0.65	54.6
			5,000	259	0.62	134.1
			7,500	253	0.60	306.8
			10,000	253	0.60	510.3
diff_eq	24	0.0	2,500	18	0.75	0.4
			5,000	18	0.75	0.7
			7,500	18	0.75	1.1
			10,000	18	0.75	1.5
Ave.	–	–	–	–	0.72	–

Table 4 MUX Cost of jpeg_dct under combination of N and $loop_{wb}$.

MUX Cost		N						
		250	500	1,000	2,500	5,000	7,500	10,000
$loop_{wb}$	100	93	93	90	88	88	88	88
	250	–	94	93	88	88	88	88
	500	–	–	92	89	89	89	89
	1,000	–	–	–	92	92	91	91
	2,000	–	–	–	91	91	91	91

are shown in **Table 4**, **Table 5**, **Table 6**, and **Table 7**. Used benchmarks were jpeg_dct and jacobi. These results show that the proposed method tends to achieve better MUX Cost at a small value of $loop_{wb}$ than at a large value of $loop_{wb}$. This is attributed to high repetitions of the massive modification of the current binding. Computation time, however, tends to increase at a small value of $loop_{wb}$. This is caused by the increase of the number of times weighted bipartite matching algorithm is performed. The value of N and $loop_{wb}$ cannot be determined uniquely. They need to be determined by considering trade-off

Table 5 Computation time of jpeg_dct under combination of N and $loop_{wb}$.

run time [s]		N						
		250	500	1,000	2,500	5,000	7,500	10,000
$loop_{wb}$	100	0.3	0.7	1.5	6.4	16.7	26.4	36.8
	250	–	0.5	1.2	5.1	14.0	23.5	32.9
	500	–	–	1.1	5.2	15.9	27.2	38.6
	1,000	–	–	–	3.3	12.6	22.0	31.3
	2,000	–	–	–	4.3	12.1	20.0	27.9

Table 6 MUX Cost of jacobi under combination of N and $loop_{wb}$.

MUX Cost		N						
		250	500	1,000	2,500	5,000	7,500	10,000
$loop_{wb}$	100	275	267	264	260	253	250	250
	250	–	285	278	273	254	249	249
	500	–	–	288	270	260	257	256
	1,000	–	–	–	272	259	253	253
	2,000	–	–	–	273	260	258	258

Table 7 Computation time of jacobi under combination of N and $loop_{wb}$.

run time [s]		N						
		250	500	1,000	2,500	5,000	7,500	10,000
$loop_{wb}$	100	7.9	14.7	29.6	99.1	310.1	505.1	734.5
	250	–	11.6	23.8	73.6	197.4	351.4	543.3
	500	–	–	22.9	67.1	198.9	407.8	615.9
	1,000	–	–	–	54.6	134.1	306.8	510.3
	2,000	–	–	–	52.6	131.7	311.7	518.8

between reduction of MUX Cost and computation time, and robustness for initial binding.

4.2 Robustness Evaluation

Our approach is based on taboo search, which does not guarantee that optimal solution can be achieved, hence robustness of the proposed method for initial binding need to be evaluated. Each operation and each variable were randomly assigned to a FU and a register, respectively in this experiment. These random binding were then used for initial binding of the proposed method.

Also, naive taboo search (NTS) was implemented to evaluate efficiency of the proposed method. A cd-neighborhood is not adopted in NTS. The neighborhoods used in NTS are generated as follows.

- A modification of the current FU binding: for each FU $f_i \in F$, move each operation $o_j \in O_b(f_i)$ to other FU $f_k(i \neq k)$ or swap every two operations bound to different FUs, if possible.
- A modification of the current register binding: for each register $r_i \in R$, move each variable $v_j \in V_b(ri)$ to other register $r_k(i \in k)$ or swap every two variables bound to different registers, if possible.

Additionally, the massive modification of the current binding is not executed.

Number of trials was five hundreds. Parameters N and $loop_{wb}$ were set to (5,000, 1,000), respectively. Other assumptions were the same as Section 4.1, and a benchmark was jpeg_dct.

The distribution of the MUX Cost for all trials is shown in **Fig. 3**. The Summary of the MUX Cost and computation time for all trials is shown in **Table 8**. Trials in which MUX Cost is up to minimum plus 5% cover 97% of all trials in the proposed method. In contrast, those of NTS cover only 46%. Also, difference between the minimum and the maximum of MUX Cost in the proposed method

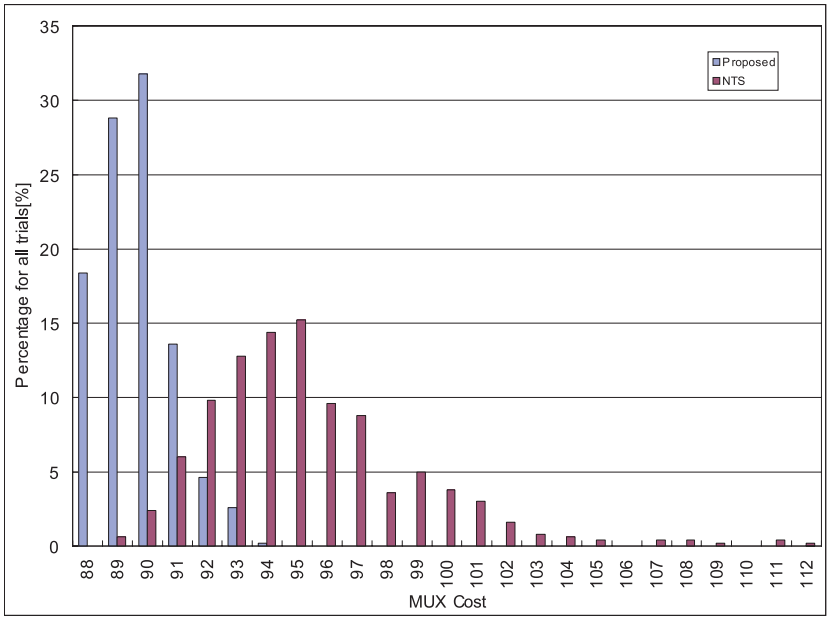

Fig. 3 Distribution of MUX Cost.

Table 8 Summaries of MUX Cost and computation time.

	MUX Cost				run time [s]		
	Min.	Ave.	Max.	Max./Min.	Min.	Ave.	Max
Initial binding	235	255.6	275	–	–	–	–
proposed	88	89.7	94	1.07	5.7	10.3	13.6
NTS	89	95.4	112	1.26	28.5	33.0	37.9

is 7%, while that of NTS is 26%. Furthermore, computation time of the proposed method is about one third of that of NTS on average. These results show that the proposed method barely depends on initial binding. Table 8 also shows that search process can be efficient by cd-neighborhood and the massive modification of the current binding.

5. Conclusions and Future Work

In this paper, we propose binding method based on an iterative improvement algorithm to optimize MUX Cost. The proposed method applies local improvement based on taboo search to FU binding and register binding iteratively. Search process can be efficient due to connection driven-neighborhoods which focus on an effect of sharing of connection. Also, stagnation of search process is avoided by the massive modification of the current FU binding and the current register binding. Experimental results show that proposed method can reduce MUX Cost by 30% on an average compared to a traditional binding algorithm. Also, computation time of the proposed method is well within utility. Additionally, the proposed method barely depends on initial binding. Advancement of robustness for initial binding and reduction of computation time, and combining modification of port assignment with binding refinement are our future work.

Acknowledgments This work has been supported by CREST-DVLSI of Japan Science and Technology Agency from 2007 to 2012 and the Grant-in-Aid for Scientific Research (B) No. 20300020 of the Japan Society for the Promotion of Science. We are grateful for their support.

References

1) Pangrle, B.: On the Complexity of Connectivity Binding, *IEEE Trans. on Computer Aided Design*, Vol.10, No.11, pp.1460–1465 (Nov. 1991).
2) Tseng, C.-J. and Siewiorek, D.P.: Automated Synthesis of Data Paths in Digital Systems, *IEEE Trans. on CAD of ICAS*, Vol.5, No.3, pp.379–395 (July 1986).
3) Paulin, P.G., Knight, J.P. and Girczyc, E.F.: HAL: A Multi-Paradigm Approach to Automatic Data Path Synthesis, *Proc. 23rd ACM/IEEE Design Automation Conference*, pp.263–270 (July 1986).
4) Pangrle, B.M.: Splicer: A Heuristic Approach to Connectivity Binding, *Proc. 25th ACM/IEEE Design Automation Conference*, pp.536–541 (June 1988).
5) Huang, C.-Y., Chen, Y.-S., Lin, Y.-L. and Hsu, Y.-C.: Data Path Allocation Based on Bipartite Weighted Matching, *Proc. 27th ACM/IEEE Design Automation Conference*, pp.499–504 (June 1990).
6) Zhu, H.W. and Jong, C.C.: Interconnection Optimization in Data Path Allocation Using Minimal Cost Maximal Flow Algorithm, *Microelectronics*, Vol.33, No.9, pp.749–59 (Sep. 2002).
7) Chen, D. and Cong, J.: Register Binding and Port Assignment for Multiplexer Optimization, *Proc. 2004 ASP-DAC*, pp.68–73 (Jan. 2004).
8) Kim, T. and Liu, C.L.: An Integrated Data Path Synthesis Algorithm Based on Network Flow Method, *Proc. IEEE Custom Integrated Circuits Conference*, pp.615–618 (May 1995).
9) Cong, J. and Xu, J.: Simultaneous FU and Register Binding Based on Network Flow Method, *Proc. Design Automation and Test in Europe*, pp.1057–1062 (Mar. 2008).
10) Choi, K. and Levitan, S.P.: A Flexible Datapath Allocation Method for Architectural Synthesis, *ACM Trans. on Design Automation of Electronic Systems*, Vol.4, No.4, pp.376–404 (Oct. 1999).
11) Avakian, A. and Ouaiss, I.: Optimizing Register Binding in FPGAs Using Simulated Annealing, *Proc. 2005 International Conference on Reconfigurable Computing and FPGAs*, p.16 (Sep. 2005).
12) Raje, S. and Bergamaschi, R.: Generalized Resource Sharing, *Proc. International Conference on Computer Aided Design*, pp.326–332 (Nov. 1997).
13) Chen, D., Cong, J. and Fan, Y.: Low-Power High-Level Synthesis for FPGA Architectures, *Proc. International Symposium on Low Power Electronics and Design*, pp.134–139 (Aug. 2003).
14) Papadimitriou, C.H. and Steiglitz, K.: Combinatorial Optimization, Prentice-Hall (1982).

(Received May 23, 2008)
(Revised August 22, 2008)
(Accepted October 9, 2008)
(Released February 17, 2009)

(Recommended by Associate Editor: *Yuichi Nakamura*)

Sho Kodama recieved his B.E. degree from Kyushu University, Fukuoka, Japan, in 2007. His research interest include high-level synthesis.

Yusuke Matsunaga received B.E., M.E. and Ph.D. degrees in Electronics and Communication Engineering from Waseda University, Tokyo, Japan, in 1985, 1987 and 1997, respectively. He joined Fujitsu Laboratories in Kawasaki, Japan, in 1987 and he has been involved in research and development of the CAD for VLSI. From October 1991 to November 1992, he has been a Visiting Industrial Fellow at the University of California, Berkeley, in the department of Electrical Engineering and Computer Sciences. He joined the faculty at Kyushu University, Japan in 2001. He is currently an associate professor of the department of Computer Science and Communication Engineering. His research interest includes logic synthesis, formal verification, high-level synthesis, automatic test pattern generation and dependable VLSI. He is a member of IEEE, IEICE and ACM.