IEEE the 8th Asian Test Symposium (ATS'99), pp. 309-314, Nov. 1999.

# A High-Level Synthesis Approach to Partial Scan Design Based on Acyclic Structure

Tomoya Takasaki† Tomoo Inoue†† Hideo Fujiwara†

†Graduate School of Information Science
Nara Institute of Science and Technology
Nara, 630-0101 Japan
{tomoya-t, fujiwara}@is.aist-nara.ac.jp

††Faculty of Information Sciences
Hiroshima City University
Hiroshima, 731-3194 Japan
tomoo@im.hiroshima-cu.ac.jp

## Abstract

*This paper presents a high-level synthesis method for testable data paths with partial scan design based on acyclic structure. For a given scheduled data flow graph, we propose a heuristic method of operational unit binding and register binding to minimize the number of scan registers for acyclic structure without sacrifice of area overhead.*

## 1. Introduction

The greater circuit density of VLSI makes testing more important and more difficult. In order to reduce the testing cost, we need to consider testability at the early stage of VLSI design. Design cost reduction of VLSI's can be achieved by considering testability at the stage of *high-level synthesis*, which synthesizes a register-transfer-level (RTL) circuit from an abstract behavioral description. We consider a method of high-level synthesis for testable data paths based on partial scan design as a method of high-level synthesis for testability.

*Partial scan design*, which replaces a part of flip-flops in a sequential circuit by scannable flip-flops (scan flip-flops), is one of the important techniques to implement an easily testable circuit with low hardware overhead. In [3] and [4], breaking feedback loops by scan flip-flops makes sequential circuits easily testable. Note that these techniques apply a test generation algorithm for sequential circuits, since the subcircuit excluding scan flip-flops (kernel circuit) has self-loops. On the other hand, some of partial scan design techniques make sequential circuits easily testable by a test generation algorithm for combinational circuits [5][6][7][8]. These techniques are *acyclic partial scan designs*, which make acyclic structure by replacing a part of registers (sets of flip-flops) in a RTL circuit by scan registers and breaking all feedback loops including self-loops by the scan registers. Our goal is to synthesize RTL data paths which minimize the number of scan registers for acyclic partial scan design.

Many techiniques have been proposed concerning high-level sythesis for testable data paths based on partial scan design. Lee et al. [9] proposed a method of synthesizing testable data paths to improve controllability/observability of registers using the user-defined number of scan registers. Potkonjak et al. [10] proposed a method to synthesize RTL designs such that all feedback loops except self-loops can be broken using a minimal number of scan registers. Fernandez et al. [12] present a register binding method for minimizing the number of scan registers required to break all feedback loops except self-loops. Mujumdar et al. [11] proposed a binding method to reduce the number of feedback loops without the use of partial scan design. Note that the above mentioned methods suppose to apply a test generation algorithm for sequential circuits and they do not break all feedback loops.

Our goal is to minimize the number of scan registers for breaking all feedback loops including self-loops so that the kernel circuit is acyclic and to apply a test generation algorithm for combinational circuits. In order to minimize the number of scan registers required for acyclic structure, we apply two strategies. One is to avoid creating self-loops by sharing operational units or registers. The other is to bind operational units and registers so that as many feedback loops pass through the same register as possible.

Section 2 presents an overview of our high-level synthesis method. We details our operational unit and register binding methods in Section 3. We present a heuristic algorithm of binding based on minimum clique partitioning in Section 4, and finally show experimental results in Section 5.

## 2. High-level synthesis flow

High-level synthesis for data paths is to transform a behavioral description, data flow graph (DFG), into a RTL data path. In this paper, as a subproblem of high-level synthesis, we focus on the binding problem to minimize the number of scan registers for ayclic structure in synthesized RTL data paths. We assumed that the scheduling of the input DFG and the allocation have been done earlier. We use the following scheduled DFG (SDFG) as the input.

**Definition 1** *Scheduled DFG (SDFG)* is a directed graph $G_{sD} = (V_D, E_D, t, s)$. $V_D$ is the set of vertex representing operations including primary inputs and outputs. $E_D \subset V_D \times V_D$ is the set of edges representing variables. $t : V_D \rightarrow \{op_1, op_2, \ldots, op_n\}$ represents the types of operations. $s : V_D \rightarrow Z^+ \cup \{0\}$ (non-negative integer) represents the control steps when operations are executed.

For the sake of simplicity, the execution delay of each operation is assumed to be one control step.

Binding procedure consists of operational unit binding and register binding. In general, it is divided into operational unit binding and register binding and they are solved separately. Here we consider a method of operational unit binding followed by register binding. For each binding, we find the minimum clique partitioning, or clique partitioning such that the number of cliques is minimum, of a compatibilty graph in order to assign operations and variables to a minimum number of operational units and registers. Each clique (complete subgraph) represents a shared operaional unit or register. Operation and register compatibility graphs are defined as follows.

If two (or more) operations are not concurrent and they can be implemented by operational units of the same type, the operations are said to be *compatible*.

**Definition 2** *Operation Compatibility Graph (OCG)* corresponding to SDFG $G_{sD}$ is an undirected graph $G_O = (V_O, E_O)$. $v \in V_O$ is the vertex corresponding to an operation in SDFG $G_{sD}$. $(u,v) \in E_O \subset V_O \times V_O$ is an edge representing the correnponding operations $u$ and $v$ are compatible.

If two (or more) variables are alive in different intervals, the variables are said to be *compatible*.

**Definition 3** *Register Compatibility Graph (RCG)* corresponding to SDFG $G_{sD}$ is an undirected graph $G_R = (V_R, E_R)$. $v \in V_R$ is the vertex corresponding to a variable in SDFG $G_{sD}$. $(u,v) \in E_R \subset V_R \times V_R$ is an edge representing the correnponding variables $u$ and $v$ are compatible.

Using the above operation and register compatibility graphs, an optimal binding can be generated by minimum clique partitioning. There may exist several equivalent solutions in terms of number of operaional units or registers, but these solutions are not necessarily equivalent in terms of scan overhead required for acyclic structure. Note that our goal is to find optimal operational unit and register bindings for acyclic partial scan design. Hence, we define weight of a clique as a necessity of scan registers, and our goal is reduced to finding the minimum clique partitioning of which weight is the smallest of all the minimum clique partitionings. In the next section, we will discuss clique weight in compatibility graphs for operational unit and register bindings.

# 3. Binding for acyclic partial scan design

## 3.1. Operational unit binding

Here we consider to reduce loops formed by sharing operations and to make easy to share scan registers required for such loops during the subsequent register binding.

If there exists a path between two compatible operations in SDFG and the operations are shared as an operational unit, then the path becomes a loop through the shared operational units in RTL data path. Therefore, at least one variable on the path is assigned to a scan register. When the length of the path or the number of variables on the path between compatible operations is large, flexibility to select a scan register is increased. In general, there exists more than one path between two compatible operations. For the sake of simplicity, we regeard the shortest path as a representative of several paths between two compatible operations. The shortest path can be considered to as a path on which a variable is the hardest to share a scan register. Besides, when the lifetimes of variables which are assigned to scan registers are long, it is difficult to share a scan register. An exact lifetime estimation is the task of register binding. Here we simply estimate the time difference of the path between two compatible operations instead of the lifetime.

From the above consideration, we give the following weight for each edge of operation compatibility graph.

### 3.1.1 Weight of operation compatibility graph

Edge $(u,v)$ weight of operation compatibility graph

(1) There exists no path between the correnponding operation $u$ and $v$ in SDFG (with either direction of $u \rightarrow v$ or $v \rightarrow u$) : $w_o(u,v) = 0$

(2) There exists a path between the correnponding operation $u$ and $v$ in SDFG : Let $p$ be the shortest path between $u$ and $v$ (when there exist paths with both directions of $u \rightarrow v$ and $v \rightarrow u$, $p$ is the shortest path among all the paths). Let $l(p)$ and $t(p)$ be the length and the time difference (time for short) of $p$, respectively,

$$w_o(u,v) = t(p) \times K_{l(p)}$$

where $K_{l(p)}$ is a sufficiently large number satisfying $K_{l(p)} > K_{l(p)+1}$.

Using the above expression, when the length of the shortest path is short, the weight becomes large. Furthermore, the time of the shortest path multiplying the weight is used to discriminate the weights of the same length of shortest paths. Among sharing pairs of the same length of shortest paths, it represents to give priority to the pairs of the shortest corresponding time.

The weight is small when the path between operations is long. Hence actually it is sufficient to estimate only sharings

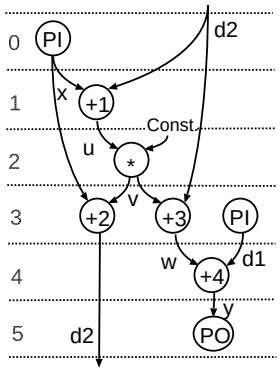
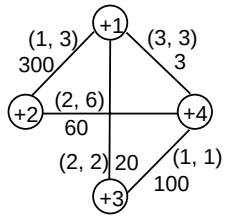



**Figure 1. Scheduled DFG $G_{sD}$**




(Length, Time)
Weight
$K_3$=1, $K_2$=10, $K_1$=100

**Figure 2. Weighted operation compatibility graph for $G_O$**

of which path is short to some extent. We estimated only paths of which length is up to 5 in experiments (See Section 5).

**Example 1** For operation compatibilty graph with addition in SDFG of Figure 1, the length and the time of the shortest path corresponding to each edge is shown in parenthesis in Figure 2. Here assumed that present operations of step 5 and next operations of step 0 are executed within the same control step. The weight for each edge is shown in Figure 2 when $K_3 = 1$, $K_2 = 10$, and $K_1 = 100$.

#### 3.1.2 Clique weight

We consider the weight of a clique as a measure representing optimality for scan of a shared operational unit. On constructing a clique by operation compatibility graph, it is desirable that the clique includes few edges with high weight. Hence we define weight of a clique as the sum of the weights of all edges in the clique.

$$W_o(C_i) = \sum_{e \in E_i} w_o(e) \quad \text{(provided clique } C_i = (V_i, E_i))$$

#### 3.1.3 Weighted minimum clique partitioning

We consider the weight of a clique partitioning for scan. It is desirable that the clique partitioning includes few cliques with high weight. Hence we define weight of a clique partitioning as the sum of the weights of all cliques in the partitioning. In order to generate an optimal operational unit binding for scan, we formulate the following problem.

Weighted Minimum Clique Partitioning Problem

**Given :** An operation compatibility graph $G_O = (V_O, E_O)$

**Solution :** A clique partitioning $\pi = (C_1, C_2, \ldots, C_n)$ such that the sum of weights of cliques $\sum_{i=1}^{n} W_o(C_i)$ is minimum (Assumed a clique $C_i = (V_i, E_i)$, $V_O = V_1 \cup V_2 \cup \ldots \cup V_n$ and $V_i \cap V_j = \emptyset$, $\forall i \neq j$)

**subject to :** the number of cliques $n$ is minimum

After applying the above clique partitioning for an operation compatibility graph, as a graph to represent sharing operational units for SDFG, create a graph merging vertices corresponding to operations to be assigned to the same operational unit as a vertex. This graph is called *operation bound graph*.

**Example 2** For weighted operation compatibility graph in Figure 2, the weighted minimum clique partitioning is $\{\{+1, +3\}, \{+2, +4\}\}$. Here the minimum sum of weights of cliques is $32 + 80 = 112$. As a result, the solution has no self-loop but the others have more than one self-loop. The operation bound graph is shown in Figure 3.

### 3.2. Register binding

Here we consider to reduce loops formed by sharing variables and to share as many scan registers required for such loops as possible.

If there exists a path between two compatible variables in operation bound graph and the variables are shared as a register, then the path becomes a loop through the shared registers. Terefore, one of the variables which include the sharing variables on the path must be assigned to a scan register. In particular, when two compatible variables adjacent in operation bound graph are shared as a register, the path becomes a self-loop through the shared register and the register must be a scan register. When there exists a path without adjacent between two sharing variables in operation bound graph, whether the shared register is assigned to a scan register depends on whether the other variables on the path are assigned to scan registers. On sharing two compatible variables as a register, We represent whether the register is assinged to a scan register as the weight for the corresponding edge of register compatibility graph.

Moreover, regardless of sharing with any other variable, variables forming self-loop in operation bound graph must be assinged to scan registers. We represent that as the weight for each vertex of register compatibility graph.

#### 3.2.1 Weight of register compatibility graph

We give the following weight for each edge of register compatibility graph.

Edge $(u, v)$ weight of register compatibility graph :

(1) There exist no path between the corresponding variable $u$ and $v$ in operation bound graph (with either direction of $u \to v$ or $v \to u$) : $w_{er}(u, v) = 0$

(2) There exists a path between the corresponding variable $u$ and $v$ in operation bound graph (with either direction of $u \to v$ or $v \to u$) :

(2-1) variable $u$ and $v$ are adjacent : $w_{er}(u, v) = 1$

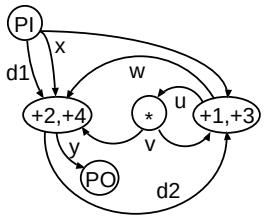



**Figure 3. Operation bound graph $G_{oD}$**

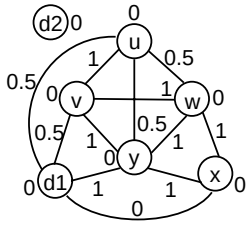



**Figure 4. Weighted register compatibility graph for $G_R$**

(2-1) variable $u$ and $v$ are not adjacent : $w_{er}(u,v) = \omega$ (provided $0 < \omega < 1$)

We give the following weight for each vertex (variable) of register compatibility graph.

Vertex $v$ weight of register compatibility graph :

(1) Corresponding variable $v$ in operation bound graph forms a self-loop : $w_{vr}(v) = 1$

(2) Otherwise : $w_{vr}(v) = 0$

**Example 3** For register compatibility graph in SDFG of Figure 1, after operational unit binding as operation bound graph of Figure 3, each edge and each vertex are weighted as shown in Figure 4 provided that ω = 0.5.

### 3.2.2 Clique weight

We consider the weight of a clique as a measure representing necessity for scan of a shared register. On constructing a clique by register compatibility graph, when the shared register (clique) includes a sharing of variables forming a self-loop (an edge with weight 1) or a variable forming a self-loop (a vertex with weight 1), the register must be assigned to a scan register. Hence we define weight of a clique as the maximum value of the weights of all edges and all vertecies in the clique.

$$W_r(C_i) = \max\{\max_{e \in E_i} w_{er}(e), \max_{v \in V_i} w_{vr}(v)\}$$

(provided clique $C_i = (V_i, E_i)$)

On sharing all the variables in a clique as a register, the weight of the clique represents whether the corresponding register is assigned to a scan register.

### 3.2.3 Weighted minimum clique partitioning

We consider the weight of a clique partitioning for scan. It is desirable that the clique partitioning includes few cliques with high weight, especially cliques with weight 1 which must be assigned to scan registers. Hence we define weight of a clique partitioning as the sum of the weights of all cliques in the partitioning. The weight represents the approximate number of scan registers required in RTL data path. In order to generate an optimal register binding for scan, we solve the weighted minimum clique partitioning problem in a similar way to operational unit binding.

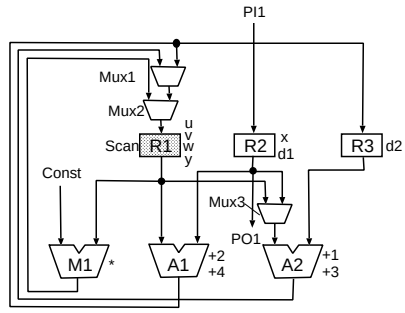



**Figure 5. Example of synthesized RTL data path**

**Example 4** For weighted register compatibility graph in Figure 4, the weighted minimum clique partitioning is $\{\{d2\}, \{u,v,w,y\}, \{d1,x\}\}$. Here the minimum sum of weights of cliques is $0+1+0=1$. At least one scan register is required by clique $\{u,v,w,y\}$ forming a self-loop. The other solutions of minimum clique partitioning require at least two scan registers because they generate two cliques forming self-loops. Hence the above cliuqe partitioning is an optimal for scan.

**Example 5** As a result of operational unit and register bindings of Example 2 and 4, the synthesized RTL data path is shown in Figure 5, and the minimum number of scan registers for acyclic partial scan design is 1. It is the minimum number of scan registers among RTL data paths synthesized from SDFG of Figure 1.

## 4. Heuristic algorithm

We present a heuristic algorithm for the weighted minimum clique partitioning problem mentioned in the previous section. This is based on a greedy algorithm [2] to find a minimum number of cliques without weights. We embed weights in the algorithm to satisfy optimality for scan while finding a minimum number of cliques. We can apply the same algorithm to both operational unit and register bindings by changing the computation of weight of a clique. The algorithm `weighted_min_clique` is shown in Figure 6.

For an operation or register compatibility graph `Gc`, `weighted_min_clique` selects an optimal clique partitioning `C_best` for scan among several equivalent solutions in terms of minimum number of cliques. This algorithm first assigns a clique to each vertex. In operation and register compatibility graphs, each vertex and edge represent a clique and a pair of sharing cliques, respectively.

In order to make a search for optimal design more widely, the algorithm repeatedly generates solutions of clique partitioning several times using

```
weighted_min_clique(Gc = (Vc, Ec, w_v, w_e))
{
  while (L < Lmax) {
    ++L;
    C = {Vc};
    /* First assign a clique to each vertex */
    (Cs1, Cs2) = select_start_pair(C);
    /* Select a start pair of sharing cliques
       on heuristic function H1 */
    merge(Cs1, Cs2);
    while(Exist a pair of sharing cliques) {
      (Ci, Cj) = select_clique_pair(C);
      /* Select a pair of sharing cliques
         on heuristic function H2 */
      merge(Ci, Cj);
    }
    if ((number(C) < number(C_best)) ||
       ((number(C) == number(C_best)) &&
        (Ws(C) < Ws(C_best)))) {
      C_best = C;
      /* Update solution */
    }
  }
}
```

- Input : compatibility graph `Gc = (Vc, Ec, w_v, w_e)`
  - `Vc` : operation/variable in SDFG
  - `Ec` : sharing possible relation between operations/variables
  - `w_e` : weight of edge
  - `w_v` : weight of vertex
- Output : optimal clique partitioning `C_best`
- `number(C)` : number of cliuqes
- `Ws(C)` : sum of weights of cliques

**Figure 6. Heuristic algorithm for weighted minimum cliue paritioning**

`select_start_pair(C)`, and select one whose sum of weights of cliques is minimum while the number of cliques is minimized among the solutions as an optimal solution. In `select_start_pair(C)`, we select a start pair of sharing cliques by a heuristic funtion $H_1$ and merge the cliques. The heuristic function gives priority to the edge (pair of sharing cliques) with smaller weight of a compatibility graph. The repetition is continued as far as we can obtain a clique partitioning such that the number of cliques is smaller or the sum of weights of cliques is smaller with the same number of cliques than before. The total repeating number `Lmax` is experimentally evaluated.

For each repetition to find a clique partitioning, we select a pair of sharing cliques, `select_clique_pair(C)`, by a heuristic funtion $H_2(h_c, h_w)$ and merge the cliques until there exists no such a pair. The heuristic function is applied for finding an clique partitioning such that the sum of weights of cliques is small while the number of cliques is minimized. Heuristic measure $h_c$ is applied for finding a minimum clique partitioning [2]. Heuristic measure $h_w$ is applied for minimizing the sum of weights of cliques and is used for selecting one among several equivalent pairs of sharing in terms of measure $h_c$. During operational unit binding, we estimate (the weight of the selecting edge of operation compatibility graph) − (the sum of the weights of the edges not to be selected by the selecting edge), and select the edge such that the value is minimum. During register binding, we estimate (the sum of the weights of the two merging vertecies of register compatibility graph) − (the maximum value of the weights of the selecting edges and the merging vertecies), and select the edge such that the value is minimum.

**Table 1. Benchmark characteristics**

| Bench. | $l$ | #PI | #PO | #Op | #Var |
|---|---|---|---|---|---|
| LWF | 5 | 2 | 1 | 5 | 7 |
| LWF.1 | | | | | |
| Tseng | 5 | 3 | 1 | 8 | 11 |
| Tseng.1 | 6 | | | | |
| Tseng.2 | | | | | |
| Paulin | 5 | 4 | 3 | 10 | 11 |
| Paulin.1 | | | | | |
| JWF | 9 | 1 | 1 | 17 | 20 |
| JWF.1 | | | | | |
| JWF.2 | | | | | |
| JWF.3 | | | | | |
| IIR | 7 | 1 | 1 | 17 | 22 |
| IIR.1 | | | | | |
| IIR.2 | 8 | | | | |
| IIR.3 | | | | | |
| EWF | 16 | 1 | 1 | 34 | 38 |
| EWF.1 | | | | | |
| EWF.2 | | | | | |

## 5. Experimental results

In order to show the effectiveness of our proposed method, we implemented the heuristic algorithm in the previous section and obtained solutions of operational unit and register bindings for some behavioral description benchmarks. The benchmarks are six types of scheduled DFG's (SDFG's), 3rd Lattice Wave Filter (LWF), Tseng, Paulin, 4th Jaumann Wave Filter (JWF), 4th IIR Cascade Filter (IIR), and 5th Elliptic Wave Filer (EWF). '.1' and so forth represent some scheduling variations. For each benchmark, latency($l$), number of primary inputs(#PI), number of primary outputs(#PO), number of operations(#Op), number of variables(#Var) are shown in Table 2.

Concerning weights of operation compatibility graph, we estimate only shortest paths of which length is up to 5. Thus the constant $K_i$ is set to $K_{l(p)} = 32^{5-l(p)}$ where $l(p)$ is the length of shortest path $p$. During register binding, the value of $\omega$ is set to be 0.5. In order to examine an appropriate value of the repeating number to find clique partitionings, the total repeating number `Lmax` is set to the maximum value or the number of all edges of each compatibility graph.

For the resultant RTL data paths, a minimum number of scan registers required for acyclic structure is obtained by using the algorithm in [13]. The experimental results are shown as ST in Table 2.

In Table 2, #OU, #Mux, #Reg and #Scan denotes the number of operational units, 2-input multiplexors, registers and scan registers in synthezied RTL data path. CPU denotes the CPU time in seconds to find clique partitionings by repeating `Lmax` times for operational unit and register bindings on a SUN Ultra30. The notation '$<0.1$' in CPU column represents that CPU time is less than 0.1 seconds.

To show the effect of weights of operation and regis-

**Table 2. Experimental results**

| Bench. | M | RTL Characteristics #OU | #Mux | #Reg | #Scan | CPU [s] |
|---|---|---|---|---|---|---|
| LWF | NT | 3 | 6 | 3 | 3 | <0.1 |
| | ST | 3 | 3 | 3 | 1 | <0.1 |
| LWF.1 | NT | 3 | 5 | 4 | 3 | <0.1 |
| | ST | 3 | 3 | 4 | 1 | <0.1 |
| Tseng | NT | 7 | 6 | 6 | 4 | <0.1 |
| | ST | 7 | 8 | 5 | 2 | <0.1 |
| Tseng.1 | NT | 6 | 8 | 6 | 4 | <0.1 |
| | ST | 6 | 7 | 6 | 3 | <0.1 |
| Tseng.2 | NT | 6 | 7 | 6 | 5 | <0.1 |
| | ST | 6 | 10 | 5 | 3 | <0.1 |
| Paulin | NT | 4 | 17 | 6 | 6 | <0.1 |
| | ST | 4 | 12 | 6 | 4 | <0.1 |
| Paulin.1 | NT | 5 | 16 | 7 | 7 | <0.1 |
| | ST | 5 | 13 | 7 | 5 | <0.1 |
| JWF | NT | 3 | 15 | 7 | 7 | <0.1 |
| | ST | 3 | 14 | 7 | 5 | <0.1 |
| JWF.1 | NT | 3 | 16 | 7 | 7 | <0.1 |
| | ST | 3 | 17 | 7 | 5 | <0.1 |
| JWF.2 | NT | 3 | 17 | 7 | 7 | <0.1 |
| | ST | 3 | 17 | 7 | 6 | <0.1 |
| JWF.3 | NT | 4 | 17 | 8 | 8 | <0.1 |
| | ST | 4 | 17 | 8 | 4 | <0.1 |
| IIR | NT | 5 | 21 | 7 | 7 | <0.1 |
| | ST | 5 | 16 | 7 | 4 | <0.1 |
| IIR.1 | NT | 5 | 23 | 7 | 7 | 1.0 |
| | ST | 5 | 20 | 7 | 4 | 1.0 |
| IIR.2 | NT | 4 | 22 | 7 | 7 | 1.0 |
| | ST | 4 | 20 | 7 | 4 | 1.0 |
| IIR.3 | NT | 4 | 21 | 7 | 7 | 1.0 |
| | ST | 4 | 13 | 7 | 4 | 1.0 |
| EWF | NT | 4 | 39 | 11 | 11 | 56.0 |
| | ST | 4 | 33 | 11 | 7 | 53.0 |
| EWF.1 | NT | 4 | 35 | 11 | 11 | 56.0 |
| | ST | 4 | 37 | 11 | 7 | 53.0 |
| EWF.2 | NT | 4 | 35 | 11 | 11 | 56.0 |
| | ST | 4 | 34 | 11 | 7 | 53.0 |

ter compatibility graphs, we also apply a non-testability method NT. The results by NT are obtained by changing the heuristic algorithm so that the resultant weights (the sums of weights of cliques) are maximum. For all benchmarks, our method ST generate a better solution than NT in terms of scan overhead. It is consider to be realized by reducing the number of self-loops and sharing scan registers effectively so that several loops pass through the same register.

As regards the repetition number of finding clique partitionings, most benchmarks can obtain the best solution by repeating less than 10 times, except large benchmarks such as EWF, about 120 times.

On the whole, applying the weighted minimum clique partitioning for operation and register compatibility graphs, it is shown that scan overhead for acyclic strucutre can be reduced and it is nearly optimal.

## 6. Conclusions

In this paper, for a scheduled behavioral description (data flow graph), we proposed a high-level synthesis method of testable register-transfer-level data paths to minimize the number of scan registers for acyclic structure without increasing the number of operational units and registers compared with previous methods without consideration to testability. Moreover, we apply the proposed method to small benchmarks and show its effectiveness. We can obtain a solution of operational unit and register binding to minimize the number of scan registers for acyclic partial scan design in synthesized RTL data paths with satisfying the minimum number of resources for small benchmarks.

We will consider a shceduling method for minimumizing number of scan registers, and study a synthesis method including controllers.

**Acknowledgement:** This work was supported in part by Semiconductor Technology Academic Research Center (STARC) under the Research Project and in part by the Minisry of Education, Science, Sports and Culture, Japan under Grant-in-Aid for Scientific Research B(2) (no.09480054). Authors would like to thank Toshimitsu Masuzawa and Michiko Inoue of Nara Institute of Science and Technology for their helpful discussion.

## References

[1] H. Fujiwara, *Logic Testing and Design for Testability*, The MIT Press, 1985.

[2] P. Michiel, U. Lauther, and P. Duzy, *The Synthesis Approach to Digital System Design*, Kluwer Acedemic Publishers, 1992.

[3] K. Cheng and V. D. Agrawal, "A partial scan method for sequential circuits with feedback," *IEEE Trans. on Comput.*, Vol. 39, No. 4, pp.544-548, April 1990.

[4] D. H. Lee and S. M. Reddy, "On determining scan flip-flops in partial-scan design approach," *Proc. Int. Conf. Computer-Aided Design*, pp.322-325, 1990.

[5] R. Gupta, R. Gupta and M. A. Breuer, "The BALLAST methodology for structured partial scan design," *IEEE Trans. on Comput.*, Vol. 39, No. 4, pp.538-544, April 1990.

[6] H. Fujiwara, S. Ohtake, and T. Takasaki, "Sequential circuit structure with combinational test generation complexity," *Trans. of IEICE* (DI), Vol. J80-D-I, No. 2, pp.155-163, February 1997 (In Japanese).

[7] T. Takasaki, T. Inoue, and H. Fujiwara, "Partial scan design methods based on internally balanced structure," *Trans. of IEICE* (DI), Vol. J81-D-I, No. 3, pp.318-327, March 1998 (In Japanese).

[8] T. Inoue, T. Hosokawa, H. Fujiwara, "An optimal time expansion model based on combinational ATPG for RT level ciruits," *Proc. IEEE the 7th Asian Test Symposium*, pp.190-197, Dec. 1998.

[9] T. C. Lee, N. K. Jha, and W. H. Wolf, "Behavioral synthesis of highly testable data paths under the non-scan and partial scan environments," *Proc. Design Automation Conf.*, pp.292-297, 1993.

[10] M. Potkonjak, S. Dey, and R. K. Roy, "Behavioral synthesis of area-efficient testable designs using interaction between hardware sharing and partial scan," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 14, No. 9, pp.1141-1154, 1995.

[11] A. Mujumdar, R. Jain, and K. Saluja, "Behavioral synthesis of testable designs," *Proc. IEEE Int. Symp. on Fault-Torelant Computing*, pp. 436-445, 1994.

[12] V. Fernandez and P. Sanchez, "Partial scan high-level synthesis," *Proc. European Design and Test Conf.*, pp.481-485, 1996.

[13] S. T. Chakradhar, A. Balakrishman, and V. D. Agrawal, "An exact algorithm for selecting partial scan design," *Proc. Design Automation Conf.*, pp.81-86, 1994.