PAPER *Special Issue on Test and Diagnosis of VLSI*

High-Level Synthesis for Weakly Testable Data Paths

Michiko INOUE[†], ***Member***, **Kenji NODA**[††], ***Nonmember***, **Takeshi HIGASHIMURA**[†††], **Toshimitsu MASUZAWA**[†], ***and*** **Hideo FUJIWARA**[†], ***Members***

SUMMARY We present a high-level synthesis scheme that considers weak testability of generated register-transfer level (RTL) data paths, as well as their area and performance. The weak testability, proposed in our previous work, is a testability measure of RTL data paths for non-scan design. In our scheme, we first extract a condition on resource sharing sufficient for weak testability from a data flow graph before synthesis, and treat the condition as design objectives in the following synthesis tasks. We propose heuristic synthesis algorithms which optimize area and the design objectives under the performance constraint.

key words: *high-level synthesis, testability, sequential ATPG, non-scan design*

1. Introduction

This paper addresses testability consideration of VLSI circuits during *high-level synthesis*. Such early consideration to testability in the design process is one of the most effective ways to reduce testing cost. High-level synthesis for testability has been investigated in the several literatures [1], [2]. They include such various testability goals as partial scan designs easily testable for sequential automatic test pattern generation (ATPG) [3]–[6], full scan designs to make combinational ATPG [7] applicable, designs for hierarchical testability [8], or designs for self testability [9]–[12]. In this paper, we propose a high-level synthesis scheme generating easily testable data paths. Our work is mainly different from the previous related works in the following two points. (1) Our target is *weak testability* which is a testability measure of register-transfer level (RTL) data paths whose target is non-scan design for sequential ATPG [13]. (2) We give testability consideration from the beginning of high-level synthesis. High-level synthesis consists of several tasks such as *scheduling* and *binding*. In most works (except for a few) on high-level synthesis for testability, testability is considered only during binding after scheduling.

Scan design is the most popular method generating easily testable design both for combinational and sequential ATPG. Full scan design guarantees high fault coverage and high fault efficiency obtained by combinational ATPG, but it makes much sacrifices of performance and area and it takes long test application time. Though partial scan design improves such disadvantages while giving up combinational ATPG, there still remains problems of area overhead and long test application time. Moreover, scan design has another disadvantage, incapability of *at-speed* testing. Maxwell et al. [14] reported that test vectors for stuck-at faults applied at speed identified more defective chips than the same test vectors applied at lower speed. Recently, some non-scan design-for-testability (DFT) techniques for RTL data paths were proposed by Dey et al. [15] and Takabatake et al. [13] The latter is our previous work, where we defined a new testability measure called *weak testability* for RTL data paths and presented a DFT technique which uses *thru module* to make the data path weakly testable. Experimental results showed the effectiveness of the proposed testability measure and the DFT technique. In this paper, we consider the weak testability *during* high-level synthesis that is a design stage earlier than RTL. This is the first high-level synthesis technique generating easily testable non-scan design for sequential ATPG.

Scheduling and binding are main tasks of high-level synthesis. Scheduling assigns operations to control steps where they are executed. Binding assigns operations to operational modules, variables and delays to registers, and data transfers to interconnection units (e.g., connection lines and multiplexors). Testability of an RTL data path much depends on the data path structure, and binding determines the structure. Therefore, many previous works consider testability during binding after scheduling. There are some few works, to the best of our knowledge, that consider testability during scheduling [3], [4], [10], [12]. In the first two works [4], [10], scheduling methods are not their major contribution. They consider some supplementary heuristic rules to support the other tasks succeeding to the scheduling. In other works, generated RTL data paths are restricted to the model called a *register file* model [3] (They succeeded in simultaneous consideration of scheduling and operational module binding in

Manuscript received November 25, 1997.

Manuscript revised February 16, 1998.

[†]The authors are with the Graduate School of Information Science, Nara Institute of Science and Technology (NAIST), Ikoma-shi, 630–0101 Japan.

[††]The author is with ULSI Systems Development Laboratories, NEC Corporation, Kawasaki-shi, 211–0011 Japan.

[†††]The author is with IBM Japan Ltd., Shiga-ken, 520–2362 Japan.

this restricted model.), or a genetic algorithm approach that attempts to optimize area, performance and testability is proposed [12]. Scheduling has a great influence on possibility of resource sharing and consequently on binding. Therefore, testability consideration from scheduling is necessary to generate testable design. In this paper, we propose a high-level synthesis scheme which generates weakly testable data paths while optimizing performance and area. This scheme includes testability analysis for data flow graph (DFG) before synthesis, and high-level synthesis tasks, scheduling and binding, which consider the weak testability. The testability analysis phase is distinctive, where we extract constraints on resource sharing from a given DFG for testability. Testability is considered as constraint on resource sharing in the succeeding high-level synthesis tasks such as scheduling and binding. To optimize the testability with such other objectives as area or performance, we take a strategy of modifying some known scheduling and binding algorithms that optimize area and performance to consider the resource sharing constraint for testability within the flexibility of them.

The rest of the paper is organized as follows. In Sect. 2, some basic definitions and a definition of weak testability are given. In Sect. 3, we propose high-level synthesis scheme that generates weakly testable data path. Experimental results appear in Sect. 4. Conclusions are given in Sect. 5.

2. Models and Testability

Our high-level synthesis scheme is applied to a data flow graph (DFG), and transforms it into an RTL data path. We first define these two models at different levels.

A DFG is a digraph $G = (V, E)$, which represents behavior of a circuit. The nodes are classified into *primary inputs*, *primary outputs*, *operations* and *delays*. A delay is used in the case where an output sequence is computed iteratively for some input sequence, where a delay represents to hold a value obtained in one iteration to use in the succeeding iterations. A directed edge $e = (u, v)$ in E represents a data flow from a node u to a node v. We call a set of edges outgoing from the same tail but not incoming to a delay a *variable*. Figure 1 shows a DFG of the 5th order IIR cascade filter. It has one primary input PI, one primary output PO, 21 operations labeled by $*$ or $+$, 5 delays $D1, D2, \cdots, D5$, and 27 variables $in, out, d1, \cdots, d5, a, b, \cdots t$.

A data path consists of *hardware elements* and *connection lines*. A hardware element is a primary input, a primary output, a register, a multiplexor, or an operational module, and a connection line connects two hardware elements with some bit width.

We assume that, for any node in a DFG, there exists a path from the node to some primary output node, and that, for any hardware element in a data path, there exists a path from the hardware element to some primary

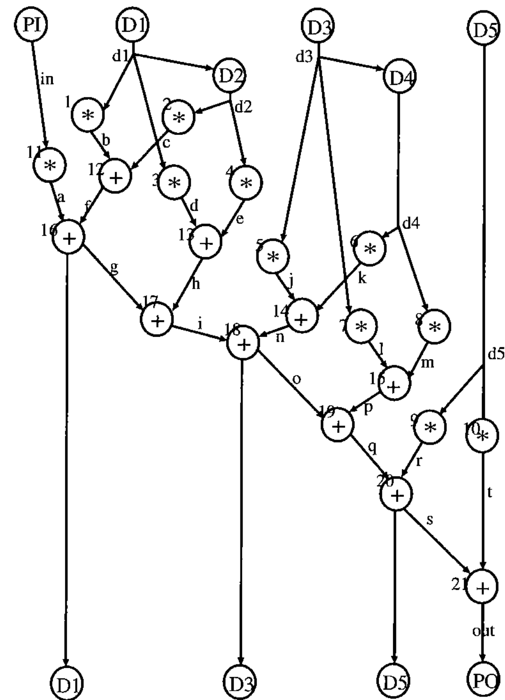


Fig. 1 DFG of the 5th IIR filter.

output.

In our previous work, we define *weak testability* as a testability measure for RTL data paths, and show its effectiveness [13].

For an operational module M, let $\mathcal{F}_M$ denote a set of operations which M provides, and $\mathcal{IN}_M$ denote a set of inputs of M. If an operational module provides an operation that returns just the value of some designated input X_i in $\mathcal{IN}_M$, such input X_i is called a *thru input*. Let $thru(X)$ denote a predicate representing that X is a thru input. Let $H_1 \rightarrow X$ (resp. $H_1 \rightarrow H_2$) mean that there is a connection line from the output of a hardware element H_1 to an input X of some hardware element (resp. some input of a hardware element H_2).

We define weak controllability and weak observability of a hardware element, and then define weak testability of a data path. Intuitively, weak controllability of a hardware element H means that *some* value (not necessarily *any*) on the output of H can be justified from primary inputs, and weak observability of a hardware element H means that *some* value on the output of H can be propagated to primary outputs. Formally these are defined as follows. For a data path, let $\mathcal{PI}$, $\mathcal{PO}$, $\mathcal{M}$, $\mathcal{R}eg$, and $\mathcal{M}ux$ denote sets of primary inputs, primary outputs, operational modules, registers, and multiplexors, respectively.

Definition 1: weak controllability [13]
A set of weakly controllable hardware elements is the minimum set $\mathcal{H}_{wc}$ satisfying the following conditions.

1. A primary input.
$I \in \mathcal{PI} \Rightarrow I \in \mathcal{H}_{wc}$.

2. A register or multiplexor with a weakly controllable input.
$H \in \mathcal{R}eg \cup \mathcal{M}ux \wedge \exists H' \in \mathcal{H}_{wc}[H' \to H]$
$\Rightarrow H \in \mathcal{H}_{wc}$.

3. An operational module only with weakly controllable inputs or with a weakly controllable thru input.
$M \in \mathcal{M} \wedge (\forall X \in \mathcal{IN}_M\ [\exists H' \in \mathcal{H}_{wc}\ [H' \to X]]$
$\vee \exists X \in \mathcal{IN}_M\ [thru(X) \wedge \exists H' \in \mathcal{H}_{wc}\ [H' \to X]]) \Rightarrow$
$M \in \mathcal{H}_{wc}$.

Definition 2: weak observability [13]
A set of weakly observable hardware elements is the minimum set $\mathcal{H}_{wo}$ satisfying the following conditions.

1. A primary output.
$O \in \mathcal{PO} \Rightarrow O \in \mathcal{H}_{wo}$.

2. A hardware element connecting to a weakly observable register or multiplexor.
$\exists H' \in \mathcal{R}eg \cup \mathcal{M}ux[H' \in \mathcal{H}_{wo} \wedge H \to H']$
$\Rightarrow H \in \mathcal{H}_{wo}$.

3. A hardware element connecting to a weakly observable operational module where the connecting input is a thru input or all the other inputs are weakly controllable.
$\exists M \in \mathcal{M}[M \in \mathcal{H}_{wo} \wedge \exists X \in \mathcal{IN}_M[H \to X \wedge$
$(thru(X) \vee \forall X' \in \mathcal{IN}_M - \{X\}[\exists H' \in \mathcal{H}_{wc}[H' \to$
$X']])] \Rightarrow H \in \mathcal{H}_{wo}$.

Definition 3: weak testability [13]
A data path DP is weakly testable iff all registers in DP are weakly controllable and weakly observable.

Since we assume that, from any hardware element, there exists a path to some primary output, it is obvious that if all registers in a data path are weakly controllable, all registers are also weakly observable, that is, the data path is weakly testable. Therefore, we consider only weakly controllability in the followings.

3. High-Level Synthesis for Weak Testability

3.1 Outline

We propose a high-level synthesis scheme that generates a weakly testable data path from a given DFG. In this scheme, we consider weak controllability of elements in a DFG in the same way as for a data path, and take a strategy of making a not weakly controllable element weakly controllable by resource sharing with a weakly controllable element. Figure 2 shows the outline of the scheme. We first analyze a DFG, and extract a sufficient condition for weak testability as a set of constraints on resource sharing. That is, if a generated data path satisfies these all constraints, it must be weakly testable and does not need any DFT for weak testability. However, such constraints on resource sharing may require more resources than the data path synthesized without the constraints. Moreover, we proposed an area efficient DFT technique for weak testability at RTL [13]. Therefore, in some case, we may obtain less area data path by generating not weakly testable data path first, and then making it weakly testable by the DFT technique afterward. From this consideration, we treat the above set of constraints on resource sharing as a *design objective* for the succeeding scheduling and binding. If generated data path is not weakly testable, we finally apply our DFT technique and obtain a weakly testable data path.

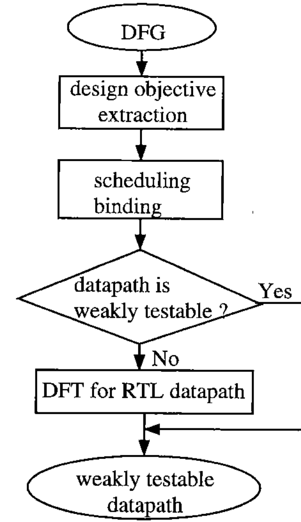


Fig. 2 High-level synthesis for weak testability.

Scheduling and binding are main tasks for high-level synthesis. Scheduling assigns operations to control steps where they are executed. Binding assigns operations to operational modules, variables and delays to registers, and data transfers to interconnection units (e.g., connection lines and multiplexors). For simplicity, we assume *disjoint operation type sets*, that is, for each operation type, a corresponding module type is uniquely determined. Moreover, we assume that all operations are *single-cycle* operations. We do not consider *multi-cycling* or *chaining*. As mentioned above, scheduling and binding consider the design objective on resource sharing for weak testability. To optimize this together with such other objectives as area or performance, we take a strategy of modifying some known high-level synthesis algorithm that optimizes such objectives as area and performance. We choose *force-directed scheduling* [16] and *left-edge register binding* [17], [18] as base algorithms to be modified.

In this paper, we define a design objective, and, for a given design objective, present scheduling, operational module binding, and register binding algorithms. Note that we just define a design objective as a sufficient condition for weak testability, but do not mention how to extract it. It is our ongoing work.

3.2 DFG Analysis

Now we consider weak testability of a data path before synthesis. First we consider weak controllability of elements in a DFG in the same way as for a data path. Intuitively, an element e in a DFG is weakly controllable if some value of e can be justified from primary inputs. Weak controllability of an element in a DFG can be also considered in a *partially* bound DFG. A partially bound DFG means that binding to hardware elements in a data path are partially determined. For such a partially bound DFG, if some element e shares the same hardware element with another weak controllable element e', we consider that e is also weakly controllable. For a partially bound DFG, we call the information to specify which elements share a hardware element a *sharing information.* Formally, a sharing information is a set $B = \{b_1, b_2, \cdots, b_k\}$ where each b_i is a set of variables and delays, or a set of the operations of the same type. We call each b_i a *sharing set.* Each sharing set b_i means that all elements in b_i share the same hardware element. We assume that sharing sets in a sharing information are disjoint, since if some element e belongs to two sharing sets b_1 and b_2 then all elements in b_1 and b_2 share the same hardware element and hence these two sharing sets should be one sharing set $b_1 \cup b_2$. We formally define weak controllability and weak testability on a DFG as follows. For a DFG, let $\mathcal{PI}_d$ denote a set of primary inputs.

Definition 4: weak controllability on a DFG
For a DFG $G = (V, E)$ and a sharing information B, a set of weakly controllable elements in G is the minimum set $\mathcal{E}_{wc}$ satisfying the following conditions.

1. A primary input.
   $pi \in \mathcal{PI}_d \Rightarrow pi \in \mathcal{E}_{wc}$.
2. An edge outgoing from a weakly controllable node.
   $(u, v) \in E \wedge u \in \mathcal{E}_{wc} \Rightarrow (u, v) \in \mathcal{E}_{wc}$.
3. A node whose all incoming edges are weakly controllable.
   $v \in V \wedge \forall u \in \{u | (u, v) \in E\}[(u, v) \in \mathcal{E}_{wc}]$
   $\Rightarrow v \in \mathcal{E}_{wc}$.
4. An edge or node which shares a hardware element with some weakly controllable one.
   $b \in B \wedge \exists x \in b[x \in E_{wc}] \Rightarrow b \subseteqq \mathcal{E}_{wc}$.

Definition 5: weak testability on a DFG
For a DFG $G = (V, E)$ and a sharing information B, if all variables and delays in G are weakly controllable, a pair (G, B) is weakly testable.

For some sharing information B, if some variable or delay is weakly controllable, the register to which it is bound is also weakly controllable if the synthesized data path P satisfies the sharing information B. If all variables and delays are weakly controllable for B, all registers in P are weakly controllable, and P is weakly testable. That is, for a DFG G and a sharing information B, if (G, B) is weakly testable, any synthesized data path satisfying B is weakly testable. In other word, if (G, B) is weakly testable, B is a sufficient condition for weak testability of a data path synthesized from G.

In the following, we call a sharing information sufficient for weak testability a *design objective* (for weak testability). A design objective DO is partitioned into two subsets *a design objective on operations* DO_f and *a design objective on variables* DO_r where DO_f is a set of sharing sets relevant to operations and DO_r is a set of sharing sets relevant to variables and delays.

3.3 Scheduling

We propose a scheduling algorithm based on *force-directed* scheduling algorithm proposed by Paulin and Knight [16]. It is a heuristic algorithm that minimizes area under time constraint, where the area is assumed to be determined (dominated) by the usage of such hardware resources as operational modules or registers. We extend it to consider both area and a design objective for weak testability.

In the force-directed algorithm, the number of hardware resources are reduced by balancing the concurrency of the operations of the same type. The algorithm is iterative, where one operation is assigned to some control step in each iteration. The selection of the operation and the control step is based on a measure called *force.* A force is calculated for each pair of an unassigned operation and a control step, and the pair with the minimum force is selected. The force reflects the influence of the assignment on the balance of the concurrency of the operations. We can use this idea for each sharing set in a design objective on operations DO_f. If we can make the concurrency of the operations in each bf_i balanced, many operations in bf_i are assigned to different control steps, and consequently, many operations can share the same operational module. In the extended scheduling algorithm, we consider a new measure *test force* as well as a *force.*

To define a *test force,* we first explain how to calculate the force. For time constraint and a (partially scheduled) DFG, a time frame of an operation is the interval $[t^S, t^L]$ where t^S and t^L and the earliest and latest control steps allowed by operation dependencies. The concurrency of the operations are computed for each operation type as expected numbers (called *distribution*) of operations assigned to each control step assuming that each operation is assigned to each control step within its time frame with the same probability.

Let an operation op be assigned to a control step s. This assignment changes not only its time frame but also the time frames of the successors and predecessors of op in the DFG. For operation o, let $tf(o)$ and $tf'(o)$ be its time frames before and after the assignment, and

$\delta(o)$ and $\delta'(o)$ be the lengths of $tf(o)$ and $tf'(o)$ (Note that $tf'(op) = [s, s]$.). Let Sc and Pr be sets of successors and predecessors of op, and let $type(o)$ denote a type of an operation o. The probability $p_o(t)$ of an operation o for a control step t, and the distribution $dist(k, t)$ of an operation type k in a control step t are defined as follows.

$$p_o(t) = \begin{cases} 1/\delta(o) & \text{if } t \text{ in } tf(o) \\ 0 & \text{otherwise} \end{cases}$$

$$dist(k, t) = \sum_{o \in \{o | type(o) = k\}} p_o(t)$$

The force $\mathcal{F}$ for the pair (op, s) is calculated as follows. To reuse this definition for consideration of the weak testability, a function $force$ takes functions $dist$ and $type$ as parameters.

$$\begin{aligned} \mathcal{F} &= force(op, s, dist, type) \\ &= \sum_{o \in \{op\} \cup Sc \cup Pr} \left\{ \frac{1}{\delta'(o)} \sum_{t \in tf'(o)} dist(type(o), t) \right. \\ &\quad \left. - \frac{1}{\delta(o)} \sum_{t \in tf(o)} dist(type(o), t) \right\} \end{aligned}$$

Let $bf_1, bf_2, \cdots, bf_m$ be sharing sets in DO_f. Now we define *sharing_type* s_type, *test distribution* $test_dist$ and *test force* $\mathcal{TF}$.

$$s_type(op) = \begin{cases} i & \text{if } op \in bf_i \\ 0 & \text{otherwise} \end{cases}$$

$$test_dist(i, t) = \begin{cases} \sum_{o \in bf_i} p_o(t) & \text{if } i > 0 \\ 0 & \text{otherwise } (i = 0) \end{cases}$$

$$\mathcal{TF} = force(op, s, test_dist, s_type)$$

In the extended scheduling algorithm, the pair of an operation and a control step with the the minimum $(\mathcal{F}, \mathcal{TF})$ in the lexicographical order is selected in each iteration. This means that we give priority a force F for area over a test force TF for testability, therefore, we can improve testability without sacrifice of area.

3.4 Binding

3.4.1 Operational Module Binding

After scheduling, operational module binding is performed. Our module binding algorithm is based on the following straightforward way: for each operational module type, select a maximal subset scheduled to distinct control steps from operations that have not been assigned, assign the operations in it to one operational module, and repeat the above selection and assignment until all operations are assigned. From the assumption that all operations are single-cycle operations, this method guarantees the binding with the minimum number of operational modules.

In our module binding algorithm, we consider the design objective within the flexibility on the selection of the maximal subset. First we estimate the number of operational modules by the above straightforward way. For each operational module type k, repeat the following assignment until all type k operations which appear in the design objective on operations DO_f are assigned.

- Select a maximum size set of unassigned operations such that all operations in it belong to the same sharing set and can be assigned to one of the estimated number of operational modules. Then, assign the operations in the set to the operational module. Here, a set of operations which can be assigned to an operational module means that all the operations in the set are scheduled to distinct control steps, and no operation scheduled to these control steps has been assigned to the operational module.

After assignment of the all operations which appear in DO_f, assign the other operations. Apply the following assignment for the operational modules one by one: select a maximal set of unassigned operations that can be assigned to the operational module and assign them. This modified method also guarantees the minimum number of operational modules.

Example: Let $op1, op2, \cdots, op9$ be the same type operations such that $\{op1, op2, op3\}$, $\{op4, op5, op6\}$ and $\{op7, op8, op9\}$ are scheduled to control steps 1, 2, and 3, respectively. Let $bf_1 = \{op1, op5, op9\}$ and $bf_2 = \{op2, op3, op4\}$ be sharing sets in DO_f. In the straightforward way, they may be assigned to three operational modules, say M_1, M_2 and M_3, as $\{op1, op4, op7\}$ to M_1, $\{op2, op5, op8\}$ to M_2 and $\{op3, op6, op9\}$ to M_3. In our algorithm, in the first iteration, each set of $\{op1, op5, op9\}$, $\{op2, op4\}$ and $\{op3, op4\}$ can be assigned to any of 3 operational modules, and the maximum one, $\{op1, op5, op9\}$, is assigned to one operational module M_1. Next, $\{op2, op4\}$ is assigned to M_2, and then $\{op3\}$ is assigned to M_3. Among the remained operations, $\{op7\}$ is assigned to M_2 and $\{op6, op8\}$ is assigned to M_3. As a result, we obtain the assignment $\{op1, op5, op9\}$ to M_1, $\{op2, op4, op7\}$ to M_2, and $\{op3, op6, op8\}$.

3.4.2 Register Binding

After scheduling and operational module binding, we perform register binding. First we assign delays to one register each. Then we assign variables to registers by a method based on a *left_edge algorithm* [17]. The left_edge algorithm is a register binding algorithm which assigns variables to the minimum number of registers. We extend this algorithm to consider the design objectives on variables DO_r.

First we explain the left_edge algorithm. For each variable in a scheduled DFG, a life time is an interval from its *birth time* to its *death time*. The birth time is the control step when the value is generated as an output of some operation, and the death time is the latest control step when the variables is referenced as an input to another operation. For two variables with lifetimes $[b_1, d_1]$ and $[b_2, d_2]$ $(b_1 \leq b_2)$, the two variables can share the same register if and only if $d_1 \leqq b_2$ holds. We briefly describe the left_edge algorithm.

1. Sort all variables in the order of the birth time.
2. Pick the first variable in the sorted list and assign it to a new register.
3. Pick the first variable in the list whose birth time are equal to or later than the death time of the last assigned variable, and assign it to the same register.
4. Repeat 3 until no variable can be assigned to the same register.
5. Repeat 2 – 4 until all variables are assigned.

In the left_edge algorithm, if there are multiple variables with the same birth time, the sort result is not unique, that is, there is some flexibility. We consider testability within this flexibility on selection of the first variable.

We modify the step 3 as follows. A variable with the smallest birth time among the variables whose birth time are equal to or later than the death time of the last assigned variable is called a *candidate variable.* If there exist multiple candidate variables, give the following rules priority. Let V_{assign} be the set of variables assigned to the current register, and let V_{cand} be the set of the candidate variables.

- If one variable in V_{assign} and another variable v in V_{cand} belong to the the same sharing set in DO_r, choose v. This has the highest priority.
- If no variable in V_{cand} belong to the same sharing set in DO_r as any variable in V_{assign}, choose a variable which does not appear in DO_r.

This modification guarantees that variables are assigned to the minimum number of registers.

Example: Let $v1, v2, \cdots v8$ be variables whose lifetimes are as in Fig. 3. Let $\{v1, v4\}$ and $\{v2, v6\}$ be sharing sets in DO_r. In the original left_edge algorithm, they may be assigned to 4 registers, say R_1, R_2, R_3, and R_4, as $\{v1, v3, v7\}$ to R_1, $\{v2, v5, v8\}$ to R_2, $\{v4\}$ to R_3, and $\{v6\}$ to R_4. In the modified algorithm, after $v1$ is assigned to R_1, $v4$ is selected among two candidate variables $v3$ and $v4$ since $v1$ and $v4$ belong to the same sharing set. As a result, we obtain the assignment $\{v1, v4, v8\}$ to R_1, $\{v2, v6\}$ to R_2, $\{v3, v8\}$ to R_3, and $\{v5\}$ to R_4.

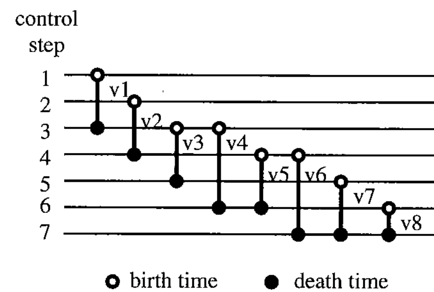


Fig. 3 Life time table.

4. Experimental Result

We made experiments on the proposed scheduling, operational module binding, and register binding algorithms. These three algorithms were applied in this order, for a given DFG and its design objective for weak testability. After register binding, we did interconnection binding using multiplexors in a straightforward way (if there are multiple connections with the same destination then realize each connection via a multiplexor), and checked the weak testability of the obtained RTL data path. If obtained RTL data path is not weakly testable, we apply the DFT technique proposed in [13] to make the data path weakly testable. In this DFT technique, thru operation is added to some inputs of operational modules.

Experiments were made on two benchmarks, the 5th order IIR cascade filter (5th IIR) and the 5th order digital elliptical filter (5th EWF), under different design objectives and different time constraints (constraints on the number of control steps). Figures 1 and 4 show DFGs of the 5th IIR and the 5th EWF. Tables 1 and 2 show the results for the 5th IIR and the 5th EWF, respectively. In each table, *time con.* denotes constraint on the number of control steps, *design objective* denotes the design objective for weak testability, *reg, add* and *mult* denote the numbers of registers, adders and multipliers of the synthesized RTL data path, and *DFT thru* denotes the number of thru inputs added to make the data path weakly testable. In addition, to show the efficiency of the weak testability, we made experiments on test generation. We applied test generation to the circuits obtained just by our high-level synthesis algorithms. We did not apply the DFT. We used a logic synthesis tool AutoLogic (Mentor Graphics Co.) and an ATPG tool TestGen (Sunrise Test System, Inc.) on a SunUltraSPARC2 (300 MHz × 2). Columns *t_eff.* and *t_gen.* denote the test efficiency and the test generation time.

We select the design objectives as follows. First, we check the weak controllability of variables and operations in a DFG. We make a sharing set by combining a weakly controllable element and not weak controllable elements with the same type. At this point, we only con-

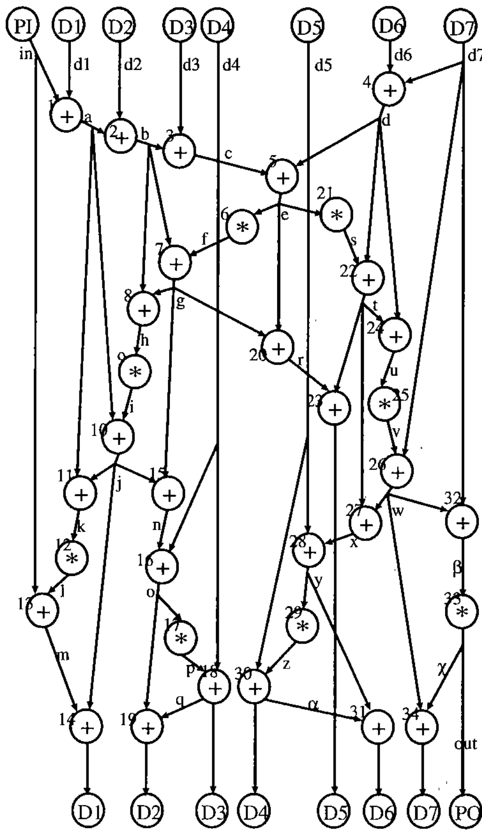


Fig. 4 DFG of the 5th EWF.

Table 1 Experimental result1: 5th IIR.

time con.	*design objective*	*data path*			*DFT*	*f_eff.*	*t_gen.*
		reg	*add*	*mult*	*thru*	[%]	[sec]
8	–	4	3	2	1	29.05	31965
	{in,f},{16,18,20}	4	2	3	0	99.88	222
	{b,f},{g,n},{11,1},{16,20}	4	2	3	1	29.98	36381
	{in,m},{p,i},{n,f},{11,7},{15,14}	4	2	3	0	99.97	426
11	–	4	2	3	1	26.26	35254
	{in,f},{16,18,20}	4	2	3	0	99.91	134
	{b,f},{g,n},{11,1},{16,20}	4	2	3	0	99.98	279
	{in,f,g},{11,9},{16,14}	4	2	3	0	99.97	66

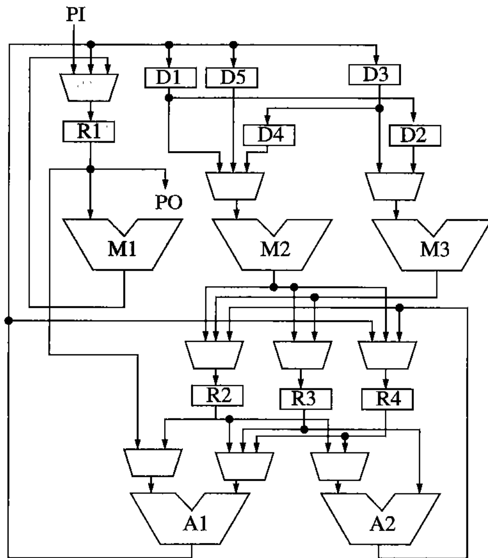
sider *mobilities* of variables and operations, where the mobility of a variable or operation means the control step interval where it can be scheduled. For example, in the case of the 5th IIR filter under a time constraint 8 (Fig. 1), a mobility of an operation 11 is $[1,2]$ and a mobility of a variable a is $[1,3]$. If two mobilities of two elements overlap, they may be scheduled to the same control step and may not be able to share the same hardware element. From this consideration, we make each sharing set from the elements with disjoint mobilities.

In both tables, the rows with no design objective

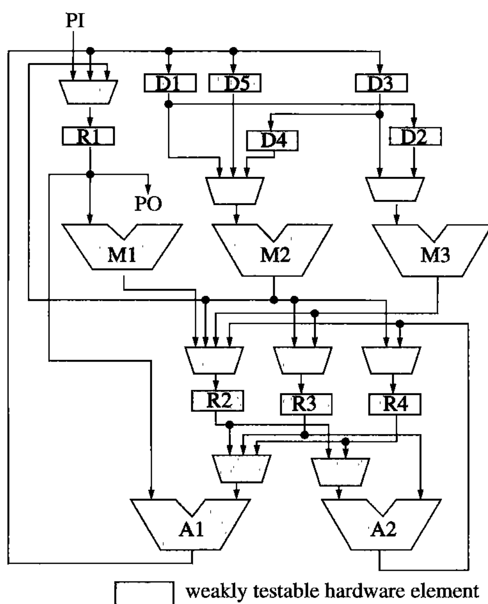
Table 2 Experimental result2: 5th EWF.

time con.	*design objective*	*data path*			*DFT*	*f_eff.*	*t_gen.*
		reg	*add*	*mult*	*thru*	[%]	[sec]
14	–	8	3	2	2	12.59	55162
	{in,y},{k,o,z},{j,q},{l,w},{p,x},{13,23,30}	8	3	2	1	11.41	63824
	{in,β},{d,w},{r,k},{13,10,15},{33,12},{1,23,30,31}	8	3	2	0	99.51	907
20	–	7	3	1	2	14.73	50487
	{in,β},{c,w},{r,k},{13,14,4},{1,2,3,30}	7	3	1	0	99.85	881
	{in,y},{k,o,z},{j,q},{l,w},{p,x},{13,23,30}	7	3	1	0	99.70	354

show the synthesized results without consideration of testability. In these cases, we did not obtain weakly testable data paths, and applied our DFT technique. Figure 5 (a) shows the data path corresponding the first row in Table 1. In this data path, only 4 hardware elements are weakly controllable before applying the DFT technique. In the cases where design objectives are considered, we can obtain weakly testable data paths without any DFT except for the two case (the third row in Table 1 and the second row in Table 2). Figure 5 (b) shows the data path corresponding the second row in Table 1. In this data path, all hardware elements are weakly controllable, therefore, the data path is weakly testable. In the case of the second row in Table 2, though we do not obtain a weakly testable data path before applying the DFT technique, we need fewer thru inputs for weak testability than the case without design objective. Moreover, the 6th row in Table 2 shows that if the time constrained is relaxed to 20, we can obtain a weakly testable data path from the same design objective without thru inputs. It is the contrast to the case without design objectives (the first row and the fourth row in Table 2). In this case, the synthesized data paths need 2 thru inputs for weak testability for both time constraints. We obtained the similar results for the 5th IIR (Table 1). The case of the third row in Table 1 need one thru input for weak testability, but we do not need any thru input for the same design objective when the time constrained is relaxed (the 7th row). We can consider that the possibility that a design objective is satisfied becomes higher when a time constraint is relaxed. In all cases with consideration of design objectives, we did not sacrifice the numbers of such resources as registers, adder or multipliers, that may dominate area of the whole circuit. Test generation results showed high fault efficiencies and short test generation times for weakly testable data paths.



(a) Synthesized data path without design objective.



(b) Synthesized data path with design objective $\{\{in, f\}, \{16, 18, 20\}\}$.

Fig. 5 Data path of the 5th IIR.

5. Conclusions

In this paper, we presented a high-level synthesis scheme that considers weak testability of generated RTL data paths, as well as their area and performance. In this scheme, we consider testability from the beginning of the high-level synthesis. This differs from most related works, in which testability is considered only during binding after scheduling.

In our scheme, first, we analyze a given DFG before synthesis and extract design objective on resource sharing for weak testability. We showed a sufficient condition of the design objective for the synthesized data path to be weakly testable. We proposed heuristic algorithms for scheduling, operational module binding and register binding which optimize area and the design objective under the performance constraint. Experimental results showed that the proposed algorithms can achieve weak testability without sacrifice of area of generated data paths. We found that some design objectives were satisfied but others were not. It is important to understand what design objective is good, that is, what design objective is easily satisfied. As an ongoing work, we are now considering the goodness of design objectives, and how to extract such a good design objective from a DFG, and from a control data flow graph.

Acknowledgment

The authors would like to thank Dr. Tomoo Inoue of Nara Institute of Science and Technology for his helpful comments. This work was supported in part by Semiconductor Technology Academic Research Center (STARC) under the Research Project and in part by the Ministry of Education, Science, Sports and Culture, Japan under the Grant-in-Aid for Scientific Research B(2) (No.09480054).

References

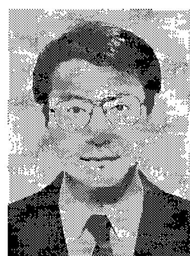
[1] K.D. Wagner and S. Dey, "High-level synthesis for testability: A survey and perspective," Proc. Design Automation Conference, pp.131–136, 1996.
[2] M.T.-C. Lee, "High-Level Test Synthesis of Digital VLSI Circuit," Artech House Publishers, 1997.
[3] M. Potkonjak, S. Dey, and R. Roy, "Considering testability at behavioral level: Use of transformation for partial scan cost minimization under timing and area constraints," IEEE Trans. Comput.-Aided-Des., vol.14, no.5, pp.531–546, 1995.
[4] T.-C. Lee, W. Wolf, and N. Jha, "Behavioral synthesis for easy testability in data path scheduling," Proc. Intl. Conf. on Computer Design, pp.616–619, 1992.
[5] T.-C. Lee, N. Jha, and W. Wolf, "Behavioral synthesis of highly testable data paths under non-scan and partial scan environments," Proc. Design Automation Conf., 1993.
[6] A. Mujumdar, R. Jain, and K. Saluja, "Incorporating performance and testability constraints during binding in high-level synthesis," IEEE Trans. Comput.-Aided-Des., vol.15, no.10, pp.1212–1225, 1996.
[7] R.B. Norwood and E.J. McClusky, "High-level synthesis for orthogonal scan," Proc. VLSI Test Symp., pp.370–375, 1997.
[8] S. Bhatia and N. Jha, "Genesis: A behavioral synthesis system for hierarchical testability," Proc. European Design and Test Conf., pp.272–276, 1994.
[9] L. Avra, "Allocation and assignment in high-level synthesis for self-testable data path," Proc. Intl. Test Conf., pp.463–472, 1991.
[10] I.G. Harris and A. Orailoğlu, "SYNCBIST: Synthesis for concurrent built-in self-testability," Proc. Intl. Conf. on Computer Design, pp.101–104, 1994.
[11] I. Parulkar, S.K. Gupta, and M.A. Breuer, "Data path al-

location for synthesizing RTL designs with low BIST area overhead," Proc. Design Automation Conf., pp.395–401, 1995.
[12] M.K. Dhodhi, I. Ahmad, and A.A. Ismaeel, "Data path synthesis for easy test testability," Proc. Asian Test Symp., pp.317–322, 1994.
[13] K. Takabatake, M. Inoue, T. Masuzawa, and H. Fujiwara, "Non-scan design for testable data paths using thru operation," Proc. Asia and South Pacific Design Automation Conf., pp.313–318, 1997.
[14] P. Maxwell, R.C. Aitken, V. Johansen, and I. Chiang, "The effect of different test sets on quality level prediction: When is 80% better than 90%?," Proc. the Intl. Test Conf., pp.358–364, 1991.
[15] S. Dey and M. Potkonjak, "Transforming behavioral specifications to facilitate synthesis of testable designs," Proc. Intl. Test Conf., pp.184–193, 1994.
[16] P. Paulin and J. Knight, "Force-directed scheduling for the behavioral synthesis of ASIC's," IEEE Trans. Comput.-Aided-Des., vol.8, no.6, pp.661–679, 1989.
[17] A. Hashimoto and J. Stevens, "Wire routing by optimizing channel assignment within large apertures," Proc. Design Automation Workshop, pp.155–159, 1971.
[18] F.J. Kurdahi and A.C. Parker, "Real: A program for REgister ALocation," Proc. Design Automation Conf., pp.210–215, 1987.

Michiko Inoue received her B.E., M.E., and Ph.D. degrees in computer science from Osaka university in 1987, 1989, and 1995 respectively. She is a research associate of Graduate School of Information Science, Nara Institute of Science and Technology (NAIST). Her research interests include distributed algorithms, parallel algorithms, and design and test of digital systems. She is a member of IEEE, IPSJ, and JSAI.

Kenji Noda received his B.E. from Kyoto university in 1995, and his M.E. from Nara Institute of Science and Technology (NAIST) in 1997. He joined NEC Corporation in 1997. His research interests include design for testability.

Takeshi Higashimura received the B.E. degree in Welding and Production Engineering from Osaka University in 1993. From 1993 to 1996, he had been engaged in the research and development of ASIC at Ricoh. Co. Ltd., Tokyo, Japan. He received the ME degree in Information Science from Nara Institute of Science and Technology (NAIST) in 1998. Since 1998 He has engaged in IBM Japan Ltd.



Toshimitsu Masuzawa received the B.E., M.E. and D.E. degrees in computer science from Osaka University in 1982, 1984 and 1987. He had worked at Education Center for Information Processing, Osaka University between 1987–1990, and had worked at Faculty of Engineering Science, Osaka University between 1990–1994. He is now an associate professor of Graduate School of Information Science, Nara Institute of Science and Technology (NAIST). He was also a visiting associate professor of Department of Computer Science, Cornell University between 1993–1994. His research interests include distributed algorithms, parallel algorithms, graph theory and design and test of digital systems. He is a member of ACM, IEEE, EATCS and the Information Processing Society of Japan.

Hideo Fujiwara received the B.E., M.E., and Ph.D. degrees in electronic engineering from Osaka University, Osaka, Japan, in 1969, 1971, and 1974, respectively. He was with Osaka University from 1974 to 1985 and Meiji University from 1985 to 1993, and joined Nara Institute of Science and Technology in 1993. In 1981 he was a Visiting Research Assistant Professor at the University of Waterloo, and in 1984 he was a Visiting Associate Professor at McGill University, Canada. Presently he is a Professor at the Graduate School of Information Science, Nara Institute of Science and Technology, Nara, Japan. His research interests are logic design, digital systems design and test, VLSI CAD and fault tolerant computing, including high-level/logic synthesis for testability, test synthesis, design for testability, built-in self-test, test pattern generation, parallel processing, and computational complexity. He is the author of Logic Testing and Design for Testability (MIT Press, 1985). He received the IECE Young Engineer Award in 1977, IEEE Computer Society Certificate of Appreciation Award in 1991, Okawa Prize for Publication in 1994, and IEEE Computer Society Meritorious Service Award in 1996. He is an advisory member of IEICE Trans. on Information and Systems and an editor of J. Electronic Testing, J. Circuits, Systems and Computers, J. VLSI Design and others. Dr. Fujiwara is a fellow of the IEEE as well as a member of the Institute of Electronics, Information and Communication Engineers of Japan and the Information Processing Society of Japan.