An Approach to Test Synthesis from Higher Level[1]

Michiko Inoue and Hideo Fujiwara

Nara Institute of Science and Technology, Ikoma, Nara 630-0101, Japan

Abstract

Consideration to testability from the early stage in the design process is one of the most effective ways to reduce testing cost. This paper addresses approaches to high-level test synthesis, especially for testability targeting non scan design. We give a brief survey of such approaches and introduce our approach. We present a high-level test synthesis method that considers testability of generated register-transfer level (RTL) data paths, as well as their area and performance. This work is mainly different from the 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. (2) We take testability into consideration from the beginning of high-level synthesis. We consider testability during both *scheduling* and *binding*, while most related works (except for a few) consider testability during binding after scheduling. We propose a heuristic synthesis algorithm that generates a weakly testable data path while minimizing area under a performance constraint.

Keywords: High-level synthesis; testability; sequential ATPG; non-scan design

1 Introduction

As the advance of VLSI technology makes circuits increasingly large and complex, testing of circuits has become more difficult and more expensive. One of the most effective ways to reduce the cost of testing is consideration to testability from the early stage in the design process. This paper addresses testability consideration of VLSI circuits during *high-level synthesis.* Since testability is one of the design criteria, high-level synthesis for testability aims to optimize testability together with other criteria such as performance, area, and power consumption.

Testing approaches are roughly classified into two types. One is external testing based on automatic test pattern generation (ATPG), and another is built-in self-testing (BIST). High-level synthesis for testability was first studied for BIST early in 1990s [1, 2, 3, 4]. High-level synthesis for ATPG based testing have also been investigated. They include various testability goals such as partial scan design [6, 7, 5, 8, 9, 10, 11, 12, 4, 13] and non-scan design [14, 15] easily testable for sequential ATPG, non-scan design to make combinational ATPG applicable [16], and designs for hierarchical testability [17, 18, 19].

Scan design is most widely used design for testability (DFT) technique. Especially, full scan design [20, 21], which replaces all flip-flops with scan flip-flops to make them completely controllable and observable, is widely accepted in the industry since combinational ATPG is applicable to such design. However, full scan design has some disadvantages: it has large area overhead, takes long test application time, and *at-speed testing* is not possible [22]. Partial scan design [23, 24], where part of flip-flops are replaced with scan flip-flops, reduces area overhead and

[1] 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).

test application time. To improve area overhead and test application time more, and to make at-speed testing possible, several non-scan DFT techniques were proposed [25, 26, 27, 28, 29, 30, 31]. This paper mainly concerned with attempts to non-scan design in high-level synthesis. First we briefly survey the researches on non scan design and its consideration in high-level synthesis. Then we introduce our high-level test synthesis method.

2 Non scan design

Scan design is a widely used DFT technique in the industry. It replaces all or part of flip-flops with scan flip-flops and connect them by one or more scan chains. In such design, scan flip-flops are controllable and observable directly from the primary inputs and at the primary outputs. Full scan design is applicable to combinational ATPG, and partial scan design reduces the difficulty of sequential ATPG. However, it has some disadvantages. It has large area overhead required to replace flip-flops with scan flip-flops. It takes long test application time and at-speed testing is not possible since test vectors are applied via the scan chains.

To resolve such problems, alternatives to scan design are investigated. Non-scan DFT techniques were first proposed to gate level sequential circuits [25, 26]. They introduce controllability and observability points to make the circuit easily testable. Dey and Potkonjak[27] presented a non-scan DFT technique for register transfer level (RTL) data paths. They defined a new testability measure, k-level testability, and proposed a method that adds test hardware to make the data path k-level testable. This measure is based on the number of control steps required to control and observe modules in a loop. These techniques aim to make the circuits easy testable for sequential ATPG.

Orthogonal scan[28] and H-SCAN[29] are DFT methods for RTL data paths. They are alternative approaches to full scan design, that is, combinational ATPG is applicable to the obtained circuits. They make all registers in a data path controllable and observable using existing paths in the circuits. These works succeeded in reducing both area overhead and test application time.

Hierarchical test generation is another approach, where test generation is divided into two phases: generating test patterns for each combinational module at gate level by combinational ATPG, and finding how to justify the patterns and propagate their responses at RT level. DFT techniques to make the RTL circuits hierarchically testable are proposed [30, 31].

3 High-level test synthesis for non scan design

3.1 High-level synthesis

High-level synthesis transforms a behavior description of a circuit into an RTL description. A behavior description is given by a hardware description language, which is first transformed into a data flow graph (DFG) or a control data flow graph (CDFG). Scheduling and binding are main tasks of high-level synthesis. Scheduling assigns operations to control steps where they are executed. Binding assigns variables to registers, operations to modules, and data transfers to interconnection units (e.g., connection lines and multiplexors). Testability of an RTL circuit much depends on its structure. Since binding determines the structure, many works consider testability during binding after scheduling. There are several works on high-level synthesis for testability targeting non scan design.

3.2 High-level synthesis for orthogonal scan

Norwood et al. considered orthogonal scan in high-level synthesis [16]. Orthogonal scan uses existing paths in a data path as scan chains. Since all registers are scanned at test mode, combinational ATPG is applicable, and hence, high fault efficiency and high fault coverage are guaranteed. Further, additional test hardware is smaller than full scan design thanks to the use of the function for normal mode, and test application time is reduced thanks to the parallel use of the bit-width of the data path. They consider orthogonal scan paths during the register binding after scheduling and module binding are applied.

Register binding can be solved as coloring on a register conflict graph which consists of variables in a scheduled DFG and conflict edges between the variables which cannot share a register because of the overlap of their lifetimes. In the register binding, first the number of registers is estimated by coloring the register conflict graph with no modification. Then it finds a scan path implementation in the scheduled DFG, where a scan path implementation is chains of variables consisting of the same number as the estimated number of registers and on which data flow at normal mode. The register conflict graph is modified by adding conflict edges between the variables on the same chain in the scan path implementation so that they are bound to the distinct registers and consequently all registers appear in the scan path. Finally, the modified register conflict graph is colored. If it fails to color within the estimated number, it backtracks to the scan path implementation. In their experiments, the proposed method obtained the designs that have as little as one-third the overhead of a traditional full scan design.

3.3 Genesis

High-level synthesis methods for hierarchical testability are also proposed [17, 18, 19]. Bhatia and Jha presented high-level data path synthesis system *Genesis* [17, 18]. Genesis performs register and module binding simultaneously for a scheduled CDFG. It consider hierarchical testability while minimizing interconnect area.

Genesis considers a test environment for each operations in CDFG. A test environment for an operation is a set of all the variables and all the operations appear on some paths in a DFG that guarantee to justify any values on the inputs from the primary inputs and to propagate any values on its output to the primary outputs. Binding uses compatibility graphs on variables and operations, where a compatibility graph consists of nodes corresponding to variables and operations and edges between two nodes that can share a register or a module. Each edge has a weight representing the preference for minimization of interconnect area. Genesis repeats the merge of two compatible nodes into a single compound node until no more merge can be done. In each iteration, Genesis merges the two compatible nodes connected by the edge with the most preferred weight among the compatible nodes whose compound node is testable if they are merged. A compound node is declared testable if one of the operations bound to the node or bound to one of its compatible nodes has test environment. If any of possible merge is not testable, extra MUX connection is added.

In experiment for some benchmarks, Genesis synthesizes the RTL circuits that are hierarchical testable and as area-efficient as the circuits synthesized other synthesis system without consideration to testability. Moreover, the obtained circuits achieved 100% fault coverage with very small test generation time.

3.4 Testability with very low area overhead

The testability goal of the orthogonal scan and Genesis is that combinational ATPG is applicable to generated circuits. Therefore, they achieve very high fault efficiency. However, there remains a problem on area overhead. Though such a goal that combinational ATPG is applicable is sufficient for high fault efficiency, it may not be necessary. We relax the testability goal and use sequential ATPG to reduce or remove area overhead, while we aim for high fault efficiency. For this purpose, we introduce *weak testability*.

4 High-level synthesis for weak testability

Now we introduce our high-level test synthesis method. Our work is mainly different from the related works in the following two points. (1) Our target is *weak testability* which is a testability measure of RTL data paths whose target is non-scan design for sequential ATPG[32]. (2) We take testability into consideration from the beginning of high-level synthesis, that is, we consider testability during both scheduling and binding. Most works on high-level synthesis for testability give consideration to testability during binding after scheduling, since testability of an RTL circuit much depends on its structure and binding determines the structure. However, 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.

4.1 Weak testability

First we give our testability goal, *weak testability* for RTL data path [32]. Intuitively, weakly testable data path guarantees that, for each hardware element, *some* value (not necessarily *any*) on the output of the element can be justified from primary inputs, and *some* value on the output of the element can be propagated to primary outputs. Though an RTL circuit consists of a data path and its controller, we assume that the controller can be modified to support such justification and propagation, and hence, we consider a data path only. We use a term *weak* since this testability does not guarantee *complete* controllability and observability. However, the experimental result shows the effectiveness of weak testability. We also define another testability measure, *weak testability cost*, to estimate the test generation time for weak testable data path.

A data path consists of *hardware elements* (primary inputs, primary outputs, registers, multiplexors, and modules)[2] and *connection lines* with some bit-width. We define weak testability of a data path using weak controllability and weak observability of hardware elements. In the following definition, $H_1 \rightarrow X$ (resp. $H_1 \rightarrow H_2$) means 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). Let $\mathcal{PI}$, $\mathcal{R}eg$, $\mathcal{M}ux$ and $\mathcal{M}$ denote sets or primary inputs, registers, multiplexors, and modules, respectively. Let $\mathcal{IN}_M$ denote a set of inputs of a module M. For an input X of a module, let $thru(X)$ denote a predicate representing that the module provides an operation that returns just the value of the X. We call such an input *thru input*. We give a definition of weak controllability only. Weak observability is defined similarly.

Definition 1 *weak controllability[32]*
A set of weakly controllable hardware elements is the minimum set $\mathcal{H}_{wc}$ satisfying the followings.

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

[2] We consider that constants are included within modules

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

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

Definition 2 *weak testability[32]*
A data path DP is weakly testable iff all registers in DP are weakly controllable and weakly observable.

We assume that, for each register, there exists a path from the register to some primary output. In this case, it is guaranteed that if all registers in a data path are weakly controllable then all registers are also weakly observable and the data path is weakly testable. Therefore, we consider only weak controllability in the followings.

We also define a measure to estimate test generation time. We define a *weak controllability cost* and a *weak observability cost* for each register and define a *weak testability cost* as sum of the weak controllability costs and the weak observability costs of all registers. To propagate a value on an input of a module to its output, it may need to justify some values on the other inputs of the module. Weak controllability cost and weak observability cost include the number of control steps required for such justification. For each hardware element H, a weak controllability cost of H, denoted by $wcc(H)$, is defined as follows.

Definition 3 *weak controllability cost [32]*

1. $wcc(I) = 0 \quad (\text{if } I \in \mathcal{PI})$

2. $wcc(R) = wcc(H) + 1 \quad (\text{if } R \in \mathcal{R}eg \wedge H \rightarrow R)$

3. $wcc(S) = \min\{wcc(H) | H \rightarrow S\} \quad (\text{if } S \in \mathcal{M}ux)$

4. $wcc(M) = \max\{wcc(H) | H \rightarrow X \wedge X \in \mathcal{IN}_M\}$
   $(\text{if } M \in \mathcal{M} \wedge \forall X \in \mathcal{IN}_M[\neg thru(X)])$

5. $wcc(M) = \min\{wcc(H) | H \rightarrow X \wedge X \in \mathcal{IN}_M \wedge thru(X)\}$
   $(\text{if } M \in \mathcal{M} \wedge \exists X \in \mathcal{IN}_M[thru(X)])$

To show the effectiveness of our definition, we made some experiments. We show the result for a RTL data path of 5th order digital elliptical filter (5th EWF, Fig.1) with the bit-width of 10. (More detailed results are seen in [32]). We compared weak testable data paths with the results obtained by other two DFT techniques. One is partial scan design to make the data path acyclic[24], another is 0-level data path proposed in [27]. To make a data path weakly testable, we make some inputs of modules thru inputs. This require small area overhead. For example, to make one input of an adder, we just add AND gates of the bit-width (Fig. 2). In this example, the value on the left input is propagated to the output when $thru = 1$. We proposed a heuristic DFT technique that add the minimum number of thru inputs to make a data path weakly testable.

The experiment used a logic synthesis tool AutoLogic (Mentor Graphics Co.), and an ATPG tool TestGen (Sunrise Test System, Inc.) on a SUN SPARCstation10. Table 1 shows the result. In the table, the columns show a type of data path (*type*), hardware overhead added to the original data-path to make it testable (*overhead*), the number of gates after logic synthesis(*#gates*),

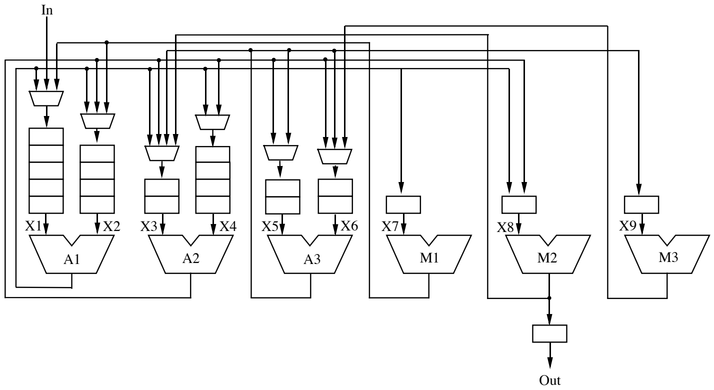
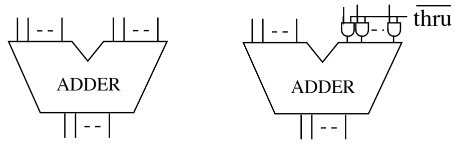


Figure 1: A data path of 5th EWF.



(a) Adder with 2 inputs (b) realization of thru input on the left input.

Figure 2: Realization of thru inputs.

the weakly testable cost(*wtc*), the number of all stuck-at faults (*#faults*), fault efficiency (*fault eff.*), the test generation time(*CPU[sec]*) and the test application time (the number of test vectors)(*appl.[cycle]*). The type *orig* denotes an original data path, LR and DP are data paths obtained by the methods in [24] and [27], respectively. Data paths added some thru inputs to make weakly testable are T1,$\cdots$,T6. In the column *overhead*, TI denotes thru input.

Our DFT technique obtained T1, where fault efficiency is increased to 97.47% from original 19.52%. As compared with LR, DP, this T1 has lower fault efficiency, but the type T9 shows that we can obtain higher fault efficiency with lower hardware overhead than LR and DP. We can also see the effectiveness of the weakly testability cost *wtc*. The result show the correlation between the weak testability cost and the test generation time.

4.2 High-level test synthesis method

4.2.1 Outline

We present a high-level test synthesis method. This is a heuristic that generates a weakly testable data path with the minimum number of resources from a DFG under a time constraint, where a resource means a module or a register, and a time constraint is given as the number of control steps in which all operations must be executed. 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.

Weak testability has a good property that, before synthesis, we can consider a condition on resource sharing sufficient for weak testability of a synthesized data path. We call such a sufficient condition *design objective for weak testability*, or just *design objective*. Figure 3 shows the outline

Table 1: Experimental result: weak testability

type	*overhead* (thru inputs)	*#gates*	*wtc*	*#faults*	*fault eff.* [%]	*CPU* [sec]	*appl.* [cycle]
orig	–	6576	–	15652	19.52	>8hr	–
LR	15 scan registers	7176	–	16252	99.46	497	19583
DP	6 multiplexors	6816	–	16132	98.12	1428	313
T1	1 TI(X1)	6586	128	15712	97.47	2116	390
T2	2 TIs(X1,X4)	6596	120	15772	98.00	1777	1054
T3	3 TIs(X1,X3,X4)	6606	116	15832	98.00	1567	905
T4	4 TIs(X1–3,X5)	6616	114	15892	98.11	1519	836
T5	4 TIs(X1,X3–5)	6616	110	15892	98.08	1512	832
T6	4 TIs(X1,X3,X4,X6)	6616	108	15892	98.16	1417	825
T7	5 TIs(X1–3,X5,X6)	6626	106	15952	98.17	1404	825
T8	6 TIs(X1–6)	6636	98	16012	98.18	1347	972
T9	9 TIs(all inputs)	6669	98	16192	99.55	1005	892

of this method. In the method, we first estimate the number of resources using *force-directed scheduling*[33] that is a heuristic minimizing the number of resources under a time constraint. Then we iteratively attempt design objective extraction, scheduling and binding until satisfying the estimation of the number of resource. In design objective extraction, we analyze a DFG and extract constraint, design objective, on resource sharing for weak testability. If we cannot obtain a data path within the predetermined iteration limit, we reduce the extracted design objectives, and iteratively attempt design objective reduction, scheduling, and binding until satisfying the resource estimation. Finally, we apply the DFT technique([32]) to a synthesized data path if it is not weakly testable.

4.2.2 Design objective

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 4 shows a DFG of the 4th order IIR cascade filter.

We define *weak controllability* of an element in a partially bound DFG like for a data path. A partially bound DFG means that binding to resources are partially determined. For a partially bound DFG, we call the information to specify which elements share a hardware element a *sharing information*. A sharing information is a set $\mathcal{B} = \{B_1, B_2, \cdots, B_k\}$ of *sharing sets*, where each sharing set B_i represents a set of elements which share the same resource. For such a partially bound DFG, if some DFG element e is weakly controllable and it shares the same resource with another DFG element e', we consider e' is also weakly controllable. We 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 [14]*
For a DFG $G = (V, E)$ and a sharing information $\mathcal{B}$, a set of weakly controllable elements in G is the minimum set $\mathcal{E}_{wc}$ satisfying the followings.

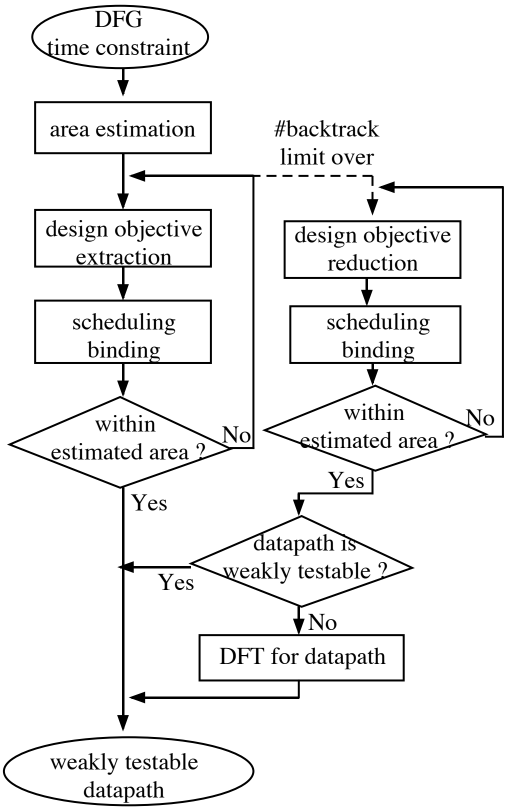


Figure 3: Outline of a high-level synthesis method.

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.
   $B \in \mathcal{B} \wedge \exists x \in B[x \in E_{wc}] \Rightarrow B \subseteq \mathcal{E}_{wc}$.

Definition 5 *weak testability on a DFG [14]*
For a DFG $G = (V, E)$ and a sharing information $\mathcal{B}$, if all variables and delays in G are weakly controllable, G is weakly testable for $\mathcal{B}$.

If some DFG element is weakly controllable for a sharing information $\mathcal{B}$, the resource to which it is bound is also weakly controllable if the synthesized data path satisfies $\mathcal{B}$. This implies that if G is weakly testable for $\mathcal{B}$, $\mathcal{B}$ is a sufficient condition for weak testability of a data path synthesized from G. A *design objective* is a sharing information sufficient for weak testability.

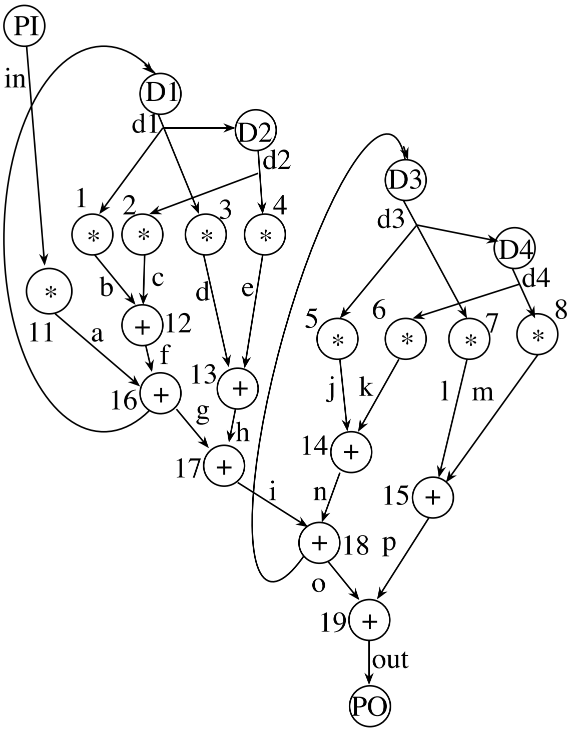


Figure 4: DFG of the 4th IIR filter.

4.2.3 Design objective extraction

In general, for a given DFG and a time constraint, there may be one or more design objectives. To extract *good* design objectives, we introduce *overlap degree* of a sharing information which represents difficulty for a sharing information to be satisfied. For two elements to share the same resource, it is necessary that these two elements do not use the same control step. For operation dependencies in a DFG and time constraint, we can find which control steps must be used, may be used, or never be used by each operation or each variable (See detailed definition in [15]). From such analysis, we derive the overlap possibility $olp(e_1, e_2)$ of two operations or two variables. The overlap possibility $olp(e_1, e_2) = 0$ implies that they never use the same control steps, therefore, they can share a resource. The value ∞ implies that they necessarily use the same control step and they cannot share a resource. The value 1 implies they may use the same control step. We define an overlap degree old of a sharing set B and of a sharing information $\mathcal{B}$ as follows.

$$old(B) = \sum_{e_1, e_2 \in B, e_1 \neq e_2} olp(e_1, e_2)$$

$$old(\mathcal{B}) = \sum_{B \in \mathcal{B}} old(B)$$

The value 0 of the overlap degree means that all elements in each sharing set are sure to use distinct control steps, while the value ∞ means some elements in some sharing set use the same control step and they cannot share a resource.

We extract a design objective with a low overlap degree from a DFG by the following greedy method. Starting from an empty sharing information, we repeatedly augment it until it becomes a design objective. We consider two types of augmentation of a sharing information $\mathcal{B}$. One is to add an element e to some sharing set B, where e is not weakly controllable for $\mathcal{B}$ and all elements in B and e are assigned to the same type of resource. Another is to add a new sharing set consisting of a weakly controllable element e_1 and a not weakly controllable element e_2 to $\mathcal{B}$, where e_2 and e_2 are assigned to the same type of resource. In each iteration, we select the next augmentation as follows.

1. Select the augmentation such that the increased overlap degree is the smallest. If there are two or more such augmentations, consider the next.

2. Select the augmentation such that the increased number of types of weakly controllable resources is the largest. If there are two or more such augmentations, consider the next.

3. Select the augmentation such that the increased number of weakly controllable elements is the largest. If there are two or more such augmentations, select one of them, arbitrarily.

An extracted design objective is considered as constraint in the succeeding high-level synthesis tasks. If the succeeding tasks fail to satisfy the extracted design objective, we extract the next design objective. In this case, the last augmentation is canceled and the design objective extraction algorithm backtracks.

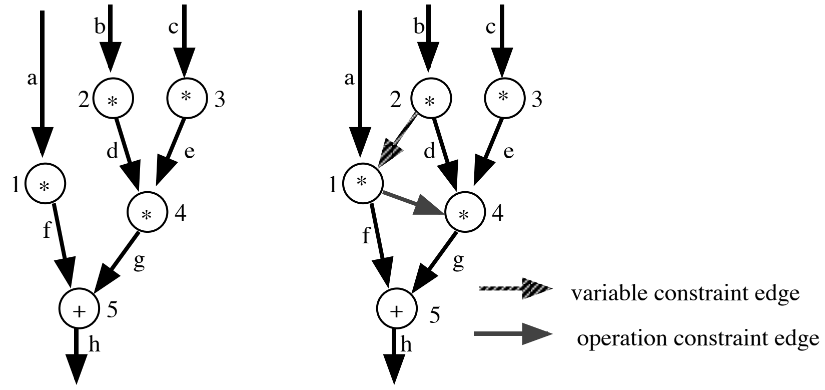
4.2.4 Scheduling

Scheduling assigns operations to control steps where they are executed. First we delete all delay nodes from a DFG, and apply a scheduling algorithm to the remained acyclic DFG. We schedule operations for a given sharing information to be satisfied. For two elements in the same sharing set to share a resource, it is necessary that two elements are assigned to different control steps. To represent such a condition, we add new types of edges to a DFG. We define two types of edges an *operation constraint edge* and a *variable constraint edge.*

An operation constraint edge (op_1, op_2) means that op_2 must be executed after op_1. Therefore, two operations never be executed at the same control step and they can share the same module. For two operations in the same sharing set with overlap possibility of 1, we add an operation constraint edge that is outgoing from the operation whose largest distance from a primary output is not smaller and incoming to the other operation. A variable constraint edge (op_1, op_2) means that op_2 must be executed at the same control step as op_1 or after. If there exist variable constraint edges from all operations that use a variable v_1 to an operation that generates a variable v_2, two variables never use the same control steps and they can share a register. For two variables in the same sharing set with overlap possibility of 1, we add variable constraint edges that are outgoing from all operations which use the variable whose largest distance from a primary output is not smaller and incoming to an operation which generates the other variable.

Example. Figure 5 shows an original DFG and an extended DFG for a design objective $\{\{1,4\},\{b,f\}\}$ where 1 and 4 are operations and b and f are variables. We add an operation constraint edge $(1,4)$ for a sharing set $\{1,4\}$ and a variable constraint edge $(2,1)$ for a sharing set $\{b,f\}$.

We schedule the above extended DFG by the modified algorithm of the *force-directed scheduling* algorithm[33]. We modified it so as to consider operation constraint edges and variable constraint edges.



(a) original DFG.

(b) extended DFG.

Figure 5: extended DFG.

4.2.5 Binding

We consider design objective during register binding and the succeeding module binding. Register binding and module binding are performed on a *register compatibility graph* and a *module compatibility graph*, respectively. These are graphs that represent which DFG elements can share a resource. In register binding, we first merge variables in the same sharing set into one node, and then apply a known minimum clique partition algorithm[34] to the merged compatibility graph. Module binding performs similarly. We first merge operations in the same sharing set into one node. Then we repeatedly pick a maximal clique from the merged compatibility graph and assign operations in the clique to one module. We repeat this until all operations are assigned. To minimize the interconnection cost, we select a maximal clique so that operations that have common input registers or common output registers belong to the same clique. Finally, we connect resources by connection lines and multiplexors according to a DFG.

4.2.6 Design objective reduction

If we cannot obtain a weakly testable data path within the iteration limit, we reduce design objectives. Design objectives obtained in the preceding extraction are reduced to sharing information that may not be sufficient for weak testability. We apply the following one element deletion and synthesis considering the reduced sharing information to the extracted design objectives in turn until we obtain a data path within the estimated number of resources. If we cannot obtain such a data path, we delete one more element from reduced sharing informations, and repeat this.

We explain how to delete one element from some sharing set in a design objective or a sharing information $\mathcal{B}$. Let G_e be an extended DFG for $\mathcal{B}$. We first find constraint edges that cause the dissatisfaction of $\mathcal{B}$. For this purpose, we apply the modified algorithm of a well known *list scheduling* algorithm[35] to G_e. It is a heuristic scheduling algorithm that minimizes the number of control steps under a resource number constraint. We modified it so as to consider constraint edges. If scheduling result exceeds the time constraint and some constraint edges appear on critical paths, we select an element to be deleted among the operations and variables corresponding to such constraint edges. Otherwise we select an element among all elements in

Table 2: Characteristic of benchmark circuits.

circuit	*#PI*	*#PO*	*#delay*	*#variable*	*#add*	*#mult.*
3rdLWF	1	1	3	7	4	1
4thIIR	1	1	4	22	8	9
4thJWF	1	1	4	20	13	4
5thEWF	1	1	7	38	26	8

Table 3: Experimental result

circuit	*time*	*design*	*data path*				*#back-*	*testability*	*fault*	*CPU*
	const.	*objective*	*#reg(v)*	*#reg(d)*	*#add*	*#mult*	*track*		*eff.[%]*	*[sec.]*
3rdLWF	4	-	4	3	2	1	-	n/wt	23.29	17754
		ex	4	3	2	1	0	wt	99.88	44
	5	-	4	3	1	1	-	n/wt	24.44	15988
		ex	4	3	1	1	0	wt	99.93	13
4thIIR	6	-	7	4	2	3	-	n/wt	24.48	36725
		ex	7	4	2	3	0	wt	99.74	385
	8	-	7	4	2	2	-	n/wt	39.35	29164
		ex	7	4	2	2	0	wt	99.88	117
4thJWF	8	-	8	4	2	2	-	n/wt	18.38	33158
		ex	8	4	2	2	1	wt	99.93	49
	11	-	8	4	2	1	-	n/wt	20.08	33120
		ex	8	4	2	1	0	wt	99.90	221
5thEWF	15	-	10	7	4	2	-	n/wt	15.11	60212
		ex	10	7	4	2	0	wt	99.38	1057
	20	-	10	7	3	1	-	n/wt	15.55	50502
		ex	10	7	3	1	0	wt	99.97	30

$\mathcal{B}$. Among these candidates, we select the element such that an overlap degree is decreased most by its deletion.

4.3 Experimental result

We made experiments on the proposed high-level test synthesis method. Experiments were made on four benchmark circuits, the 3rd order lattice wave filter (3rd LWF), the 4th order IIR cascade filter (4th IIR, Fig.4), the 4th order Jaumann wave filter (4thJWF) and the 5th order digital elliptical filter (5th EWF), under different time constraints. Table 2 shows the characteristic of these benchmark circuits. We applied our synthesis method to these and generated weakly testable data paths (Table 3). A column *time const.* denotes time constraints. In a column *design objective*, '–' means the synthesis without design objective extraction and "ex" means the synthesis considering extracted design objectives. Columns *#reg(v)*, *#reg(d)* *#add* and *#mult* denote the numbers of registers for variables, registers for delays, adders and multipliers of the synthesized RTL data path, respectively, and *#backtrack* denote the number of backtracks of design objective extraction caused by the dissatisfaction of the succeeding synthesis. In a column *testability*, "wt" means that an obtained data path is weakly testable and "n/wt" means a data path is not weakly testable. We then applied test generation to show the effectiveness of weak testability. We used a logic synthesis tool AutoLogicII (Mentor Graphics Co.) and an ATPG tool TestGen (Sunrise Test System, Inc.) on a Sun Ultra (300MHz× 2). Columns *fault eff.* and *CPU* denote the fault efficiency and the test generation time.

For all benchmark circuits and all time constraints, we obtained weakly testable data paths with the same number of resources as the case without design objective extraction. In the case without design objective extraction, we did not obtain any weakly testable data path. That is, our method realized weak testability without sacrifice of the number on resources. Moreover, in most cases (except for one case), the synthesis algorithm generated a data path for the design objective extracted first. This implies the effectiveness of the overlap degree. In the test generation result, weakly testable data paths have almost complete fault efficiency with small test generation time, while not weakly testable data paths have low fault efficiency.

5 Conclusions

Consideration to testability from higher level is one of the most effective ways to reduce the cost of testing. Though scan design is widely used in the industry, it has disadvantages to be improved and resolved. DFT and SFT(synthesis for testability) based on non scan design are important technologies. In this paper, we considered approaches to non scan design, and introduced our high-level test synthesis method.

Our high-level test synthesis method considers weak testability whose target is non-scan design for sequential ATPG. Moreover, we take testability into consideration from the beginning of high-level synthesis. We proposed a heuristic method that generates a weakly testable data path with the minimum number of resources from a DFG under a time constraint. Experimental results show that our method obtained weakly testable data paths with the same number of resources as the case without consideration to testability. That is, we achieved weak testability without sacrifice of the number of resources.

There are some remained works. One is on scalability. Though we obtained effective results for some benchmarks, we do not know the method is applicable to larger scale circuits. For larger scale circuits, we may need to consider other testability measures such as weak testability cost, or testability to make combinational ATPG applicable. Another problem is on controller. We just assume that the controller can be modified to support testing of a data path. We must consider how to modify it, and how to test the whole circuit including the controller itself.

References

[1] L. Avra, Allocation and assinment in high-level synthesis for self-testable data path, in: *Proc. Int. Test Conf.* (1991) 463–472.

[2] I. G. Harris and A. Orailoğlu, SYNCBIST: synthesis for concurrent built-in self-testability, in: *Proc. Int. Conf. on Computer Design* (1994) 101–104.

[3] I. Parulkar, S. K. Gupta and M. A. Breuer, Data path allocation for synthesizing RTL designs with low BIST area overhead, in: *Proc. Design Automation Conf.* (1995) 395–401.

[4] M. K. Dhodhi, I. Ahmad and A. A. Ismaeel, Data path synthesis for easy test testability, in: *Proc. Asian Test Symp.* (1994) 317–322.

[5] T.-C. Lee, N. Jha and W. Wolf, A conditional resource sharing method for behavioral synthesis of highly testable data paths, in: *Proc. Int. Test Conf.* (1993) 749–753.

[6] T.-C. Lee, W. Wolf and N. Jha, Behavioral synthesis for easy testability in data path scheduling, in: *Proc. Int. Conf. on Computer-Aided Design* (1992) 616–619.

[7] T.-C. Lee, W. Wolf, N. Jha and J. Acken, Behavioral synthesis for easy testability in data path allocation, in: *Proc. Int. Conf. on Computer Design* (1992) 29–32.

[8] Dey, Potkonjak and Roy, Exploiting hardware sharing in high-level synthesis for partial scan optimization, in: *Proc. Int. Conf. on Computer-Aided Design* (1993)20–25.

[9] Dey, Potkonjak and Roy, Synthesizing designs with low-cardinality minimum feedback vertex set for partial scan application, in: *Proc. VLSI Test Symp.* (1994)2–7.

[10] M. Potkonjak, S. Dey and R. Roy, Behavioral synthesis of area-effiient testable designs using interaction between hardware sharing and partial scan, *IEEE Trans. on Computer-Aided-Design* **14** (5) (1995) 1141–1154.

[11] A. Mujumdar, R. Jain and K. Saluja, Incorporationg performance and testability constraints during binding in high-level synthesis, *IEEE Trans. on Computer-Aided-Design* **15** (10) (1996) 1212–1225.

[12] L.-R. Huang, J.-Y. Jou, S.-Y. Kuo and W.-B. Lian, Easily testable data path allocation using input/output registers, in: *Proc. Asian Test Symp.* (1996) 142–147.

[13] V. Fernandez and P. Sanchez, Partial scan high-level synthesis, in: *Proc. European Design and Test Conf.* (1996).

[14] M. Inoue, K. Noda, T. Masuzawa and H. Fujiwara, High-level synthesis for weakly testable data paths, *IEICE Trans. on Information and Systems* (1998, to appear).

[15] M. Inoue, T. Higashimura, K. Noda, T. Masuzawa and H. Fujiwara, A high-level synthesis method for weakly testable data paths, in : *Proc. Asian Test Symp.* (1998, to appear).

[16] R. B. Norwood and E. J. McClusky, High-level synthesis for orthogonal scan, in: *Proc. VLSI Test Symp.* (1997) 370–375.

[17] S. Bhatia and N. Jha, Genesis: A behavioral synthesis system for hierachical testability, in: *Proc. European Design and Test Conf.* (1994) 272–276.

[18] S. Bhatia and N. Jha, Behavioral synthesis for hierachical testability of controller/data path circuits with conditinal branches, in: *Proc. Int. Conf. on Computer Design* (1994) 91–96.

[19] M. L. Flottes, Hammad and B.Rouseyre, High-level synthesis for easy testability, in: *Proc. European Design and Test Conf.* (1995)198–206.

[20] E. B. Eichelberger and T. W. Williams, A logic design structure for LSI testability, *Journal of Design Automation and Fault-Tolerant Computing* **2** (2) (1978) 165–178.

[21] M. Williams and J.B.Angell, Enhancing testability of large scale integrated circuits via test points and additional logic, *IEEE Trans. on Computers* **C-22** (1) (1973) 46–60.

[22] P. C. 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%?, in: *Proc. Int. Test Conf.* (1991) 358–364.

[23] S. T. Chakradhar, A. Balakrishnan and V. D. Agrawal, An exact algorithm for selecting partial scan flip-flops, *Journal of Electronic Testing: Theory and Applications* **7** (1995) 83–93.

[24] D. H. Lee and S. M. Reddy, On determining scan flip-flops in partial-scan design approach, in: *Proc. Int. Conf. on Computer-Aided Design* (1990) 322–325.

[25] E. M. Rudnick, V. Chickermane and J. H. Patel, Probe point insertion for at-speed test, in: *Proc. VLSI Test Symp.* (1992) 223–228.

[26] V. Chickermaned, E. M. Rudnick, P. Banerjee and J. H. Patel, Non-scan design-for-testability techniques for sequential circuits, in: *Proc. Design Automation Conf.* (1993) 236–241.

[27] S. Dey and M. Potkonjak, Non-scan design-for-testability of RT-level data paths, in: *Proc. Int. Conf. on Computer-Aided Design* (1994) 640–645.

[28] R. B. Norwood and E. J. McCluskey, Orthogonal scan: Low overhead scan for data paths, in: *Proc. Int. Test Conf.* (1996) 659–668.

[29] S. Bhattacharya and S. Dey, H-scan: A high level alternative to full-scan testing with reduced area and test application overheads, in: *Proc. VLSI Test Symp.* (1996) 74–80.

[30] I. Ghosh, A.Raghunathan and N. K. Jha, Design for hierachical testability of RTL circuits obtained by behavioral synthesis, in: *Proc. Int. Conf. on Computer Design* (1995) 173–179.

[31] I. Ghosh, A.Raghunathan and N. K. Jha, A design for testability tequnique for rtl circuits using control/data flow extraction, in: *Proc. Int. Conf. on Computer Design* (1996) 329–336.

[32] K. Takabatake, M. Inoue, T. Masuzawa and H. Fujiwara, Non-scan design for testable data paths using thru operation, in: *Proc. Asia and South Pacific Design Automation Conf.* (1997) 313–318.

[33] P. Paulin and J. Knight, Force-directed scheduling for the behavioral synthesis of ASIC's, *IEEE Trans. on Computer-Aided-Design* **8** (6) (1989) 661–679.

[34] C. Tseng and D. P. Siewiorek, Automated synthesis of data paths in digital systems, *IEEE Trans. on Computer-Aided-Design* **5** (3) (1986).

[35] M. C. McFarland, A. C. Parker and R. Camposano, Tutorial on high-level synthesis, in: *Proc. Design Automation Conf.* (1988) 330–336.

Michiko Inoue received her B.E., M.E, and Ph.D degrees from Osaka University in 1987, 1989, and 1995 respectively. She is an instructor of Graduate School of Information Science, Nara institute of Science and Technology (NAIST). Her research interests include distributed algorithms, parallel algorithms, graph theory and design and test of digital systems. She is a member of IEEE, the Institute of Electronics, Information and Communication Engineers of Japan, the Information Processing Society of Japan, and Japanese Society for Artificial Intelligence.

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 sysnthesis 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 IEEE Trans. on Computers, J. Electronic Testing, J. Circuits, Systems and Computers, J. VLSI Design and others. Dr. Fujiwara is a fellow of the IEEE and a Golden Core member of the IEEE Computer Society as well as a member of the Institute of Electronics, Information and Communication Engineers of Japan and the Information Processing Society of Japan.