High–Level Synthesis for Easy Testability

M.L. Flottes, D. Hammad, B. Rouzeyre

Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier, U.M. CNRS 9928
161 rue Ada, 34392 Montpellier Cedex 5, FRANCE

Abstract : *This paper presents an attempt towards design quality improvement by incorporation of testability features during datapath high–level synthesis. This method is based on the use of hardware sharing possibilities to improve the testability of the circuit without a time consuming re–synthesis process. This is achieved by incorporating test constraints during register allocation and interconnect network generation. The main features of this method are:*
– a test analysis at the behavioral level rather than at a structural one.
– the non limitation on the behavioral descriptions (loops, control constructs are supported),
– the optimized test area overhead and cpu time compared to standard approach,
The method was applied to several benchmarks resulting in easily testable designs for almost the same area costs as the original (without testability) designs.

I Introduction

Recent works have proved the efficiency of considering testability as one of the design constraints [2][10][14][20] during High Level Synthesis. Based on the observation that the behavioral description of a circuit can be implemented by various structures with the same area and time costs but with different testability costs, one of the goals of ''High Level Test Synthesis" is to take advantage of the design alternatives offered by high level synthesis in order to improve or guarantee the testability of the circuit.

In this paper, we present a testability driven behavioral synthesis chain, the aim of which is to generate an easily testable design (where all nodes can be easily controlled from primary inputs and easily observed from primary outputs) while keeping area overhead to a minimum. The three main tasks of high level synthesis are scheduling, allocation and generation of interconnections. Scheduling is the assignation of operations to control steps; allocation is the step during which operations are mapped to functional modules and variables to memory modules; finally, interconnections are generated between the previously defined structural entities according to the functionality of the circuit. In the presented system, register allocation and interconnect generation algorithms take into account the testability of the potential design as well as area considerations. To realize this goal, the partial design's testability is analyzed before each one of these two synthesis steps leading to the generation of constraints that are taken into account to guide the architectural choices.

We consider here architectures composed of a controller and a datapath. We deal with the datapath testability improvement assuming that the controller can be modified to support the test plan and to be self–testable [12]. Next section relates relevant works in this area. Section III is an overview of our test synthesis method while section IV gives some definitions. In section V, we present the register allocation for testability method and give some comparisons with the standard approach. The interconnect generation method is detailed in section VI.

II Related works

The traditional testability approaches, which consist in constructing a datapath without testability considerations and adding test hardware at the RT level(e.g. [6]), have two major drawbacks. Firstly, the problem of finding a minimal set of test points to make the design testable is NP–complete. Secondly, the area overhead resulting from the additional test material may be large (and is at least not under control). The earliest approaches to high level synthesis for testability [1][9] used a testability insertion back–end where a post synthesis module is added to the synthesis chain. This module converts an RTL design into a testable one by test hardware insertion. However, these techniques do not actually support testability on the behavioral level.

In [10], Gebotys et al. proposed a design and test synthesis methodology where test cost is taken into account as one of the design parameters for scan and BIST test strategies. Papachristou et al. [17] proposed a high level synthesis for testability method based on BIST insertion where allocation techniques are used to minimize the area overhead due to BIST resources. Another scheme for synthesis of BISTed data paths was proposed by Avra [2] where the number of CBILBO registers is minimized by imposing testability constraints during allocation.

Chen et al. [4] proposed a high–level testability analysis system which identifies controllable and observable nodes on a Control Flow Graph and finds test paths for them. However, only control sequences existing in the normal behavior are authorized for building up test paths (i.e. all other structural paths in the circuit, are ignored). For those nodes that are identified as hard to test, the authors propose the modification of the control graph by test point insertion (additional connections), or test statements insertion (modification of the controller).

Considering external testing, Lee et al. [14][15] have proposed two rules for synthesis for testability that are used to guide datapath allocation and datapath scheduling, which are: i) whenever possible, allocate a register to at least one primary input or primary output variable, ii) reduce the sequential depth from a controllable register to an observable register. The register allocation scheme uses the left edge algorithm while incorporating these two rules. This method is intrinsically limited to purely sequential specifications or under the hypothesis that all variables alive at the first (resp. last) c–step of a control construct (and only these ones) are connected to primary inputs (resp. outputs). As far as we are concerned, this method is inapplicable to control–dominated circuits (i.e. with specifications embedding control constructs), and functional unit transparency is not questioned.

Another proposal was made by Bhatia et Jha [3], which involves not only register allocation but also functional module allocation. They propose to map a controllable (observable) variable to an uncontrollable (unobservable) register or a testable operation to an untestable module thus making the register or module testable. This allocation scheme uses backtracking to undo allocation when a module is declared untestable and when allocation is unable to resolve a testability problem, an extra multiplexed connection is added to enhance controllability (and/or observability).

III Contribution

The method presented here overcomes the limitations of the above approaches and of a late testability insertion by providing early diagnosis of hard to test points and solutions for these testability problems. It addresses external test and aims to generate directly RTL designs where every node possesses 1/ a justification path from a primary input allowing to set it to any desired value and 2/ an observation path allowing to detect on a primary output, any error in the value carried by the node. As no assumption is made on the actual values of test data, the test paths are such that any test data can be propagated to and from any fault location. These justification and sensitization paths are composed of a cascade of transparent modules and are determined using data transfers issued from the initial behavioral description. These modules are the physical entities building up the datapath: functional units (FUs), registers, ...

Our method differs from the approach presented in [3] by using functional unit transparency properties and on the other hand, by not restricting data transfers to the behavioral paths of the normal mode of execution. Taking advantage of the alternatives of high level synthesis, we either guarantee the testability of the datapath without a high area overhead and without the time consuming backtracking step, or generate the most testable design for a given area.

Fig. 1 presents the synoptic of the complete high level

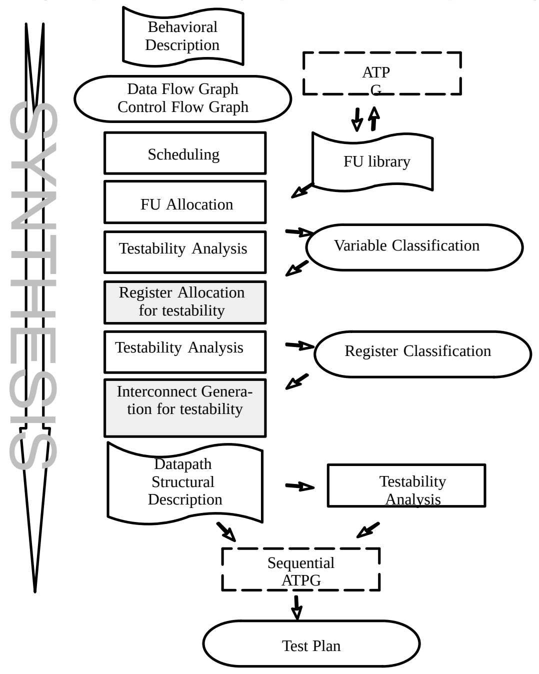

Fig. 1 Synthesis of testable datapath

test synthesis system [7]. The test synthesis methodology can be summarized as follows:

1. All justification and propagation paths for all nodes are found and all uncontrollable and unobservable nodes are detected. This testability analysis is performed on the circuit's partially determined structure, i.e. composed of FUs and variables.
2. Register allocation for testability is performed based on the information provided by the first step in order to make as many registers as possible testable.
3. Testability analysis is run again on the new partial structure now composed of FUs, registers and RAMS.
4. Interconnection generation is conducted based on the information provided by the third step.

It must be stressed that the justification and propagation behavioral paths are data–flow graphs and not structural paths. These paths are built up from the elementary data transfers of the behavioral specification, that is, they are set independently of the schedule of the specification and of the initial instruction set (we assume that the controller can be modified to support the test plan).

Next section gives some definitions used in the proposed high level test synthesis method.

IV Definitions

The testability analysis (or test path search) method is based on the following definitions:

1. Independently of its use in a circuit, a module is **C–transparent** if its output port can be set to any desired value. This property is an intrinsic property. Registers, variables, busses, muxes are C–transparent.
2. In the same way, a module is **O–transparent** if any change on the value at one of its inputs is reflected by a change at the output (without any change on the other inputs). Registers, variables, busses, muxes are O–transparent.
3. A **C–path** of an n–bitwidth variable v is a path that allows v to be assigned to the 2^n possible values from a primary input port. A C–path is composed of cascades of C–transparent modules.
4. An **O–path** of v is a path that allows any change in the value carried by v to be propagated from v to a primary output. An O–path is composed of a cascade of O–transparent modules.
5. If a variable v possesses at least one C–path, it is said to be **controllable**.
6. If a variable v possesses at least one O–path, it is said to be **observable**.
7. An **input port** of a FU is **controllable,** if at least one of the variables feeding this port is controllable.
8. A FU is **fully controllable** if each of its input ports is controllable independently.
9. An **output port** of an FU f is **observable for a fault in f**, if there exists an observable variable fed by this port such that its O–path and its lateral C–paths (C–paths which control the propagation through the O–path) do not contain f.

In our system, the transparency properties of FUs are assumed as known and are attributes in the library. C–paths and O–paths are a generalization of the notions of I–path from [1] and of S–path, F–path from [8].

The output output is the set of all C–paths (O–paths) for all controllable (observable) modules' ports (FUs, variables, registers..).

This information is used during the next synthesis steps to establish merging preferences during register allocation and interconnect generation for testability. The interested reader can refer to [11] for the exact definitions of controllability and observability as well as for the analysis process.

To illustrate the testability analysis methodology, we applied it to the differential equation benchmark from [18]. Its scheduled and partially allocated data flow graph is depicted in Fig. 2. This representation, classical in HLS is

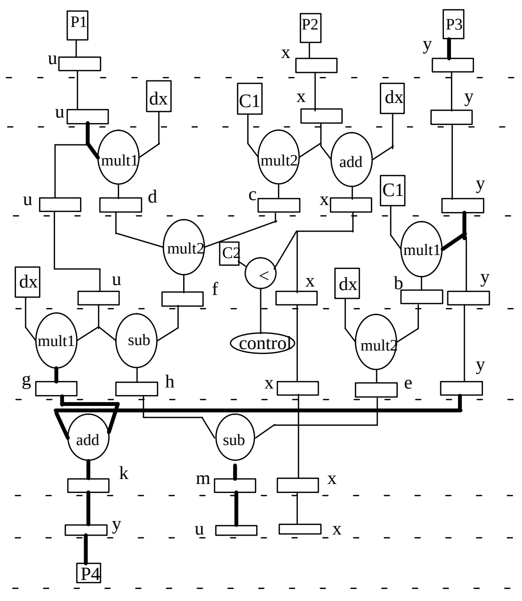

Fig. 2: Scheduled data flow graph

the system's internal representation of the behavioral description. We consider that variables x, y and u are loaded from primary inputs in the first iteration and then loaded from the FUs in the other iterations. For simplicity, the output of the comparator which communicates directly with the controller is not considered. For instance, let's assume that the 5 FUs used in the design are C– and O–transparent except the sub one. Table 1 shows the results of the testability analysis. Rows 2 and 3 give respectively the controllability and observability of the variables. Table 2 gives the number of possible justification and propagation paths for test vectors of faults occurring in the FUs.

var.	y	x	u	b	d	k	e	f	c	h	m	g
control.	c	c	c	c	c	c	c	c	c	nc	nc	c
observ.	o	o	o	no	no	o	no	no	no	no	o	o

Table1:Variables controlability / observability

	Mult1	Mult2	Add	Sub
input 1 (just. paths)	1	1	2	1
input 2 (just. paths)	1	1	1	4
output (prop. paths)	1	0	1	1

Table 2: # of justification and propagation paths for faults occurring in the FUs

Fig. 3 represents the observability path of the sub FU. It is built up from the elementary data transfers depicted by thick lines in Fig. 2.

V Register allocation for testability

The basic principle of register allocation for testability is to merge uncontrollable (unobservable) variables with controllable (observable) ones into registers in order to make these registers and the FUs connected to them controllable (observable).

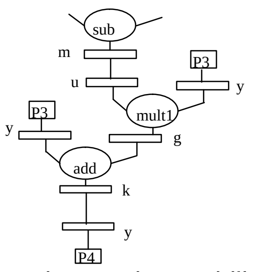

Fig. 3 : "sub" FU observability path.

Let's first recall the basic principle of register allocation: variables can share the same register if their life times do not overlap. The merging possibilities of variables are extracted performing a life time analysis and a compatibility graph is built up. A solution minimizing area expressed by a cost function f_a (nb. of registers, nb. of multiplexer inputs,...) is sought by finding an adequate clique partitioning in the compatibility graph. Generally, the search is guided by weighting the graph edges by the area gain Δf_a, i.e. the area gain due to the merge of the two involved variables (see [19] for more details). The pair of variables joined by the edge of maximal weight is merged. After each merge, all weights are updated and the process is iterated until no merge is possible.

To take account for testability, we simply replace Δf_a by $\alpha\Delta f_a / \Delta f_{a\ max} + \beta\Delta f_c/\Delta f_{c\ max} + \delta\Delta f_o/\Delta f_{o\ max}$ where Δf_c (resp. Δf_o) is the controllability (resp. observ.) gain and $\Delta f_{a\ max}$, $\Delta f_{c\ max}$ and $\Delta f_{o\ max}$ are normalization factors. α,β and δ are user defined tuning factors allowing trade–offs between area, controllability and observability. Next, we only detail the calculations of Δf_c and Δf_o can be obtained in a similar fashion.

V.1 Controllability gain Δf_c

The controllability gain is intended to measure the effect of a merge on the controllability of the whole circuit. As mentioned earlier, merging an uncontrollable variable with a controllable one into a register makes this register and the FU port connected to it controllable. Therefore, controllability gain is composed of three terms representing: i) The number of FUs that become fully controllable due to the merge, ii) the number of FU ports that become controllable, and iii) the variable merging preference. As testability bottlenecks usually concern FUs, since more test patterns are required to test a FU than to test a register , we give a priority to merges that increase the controllability of FUs over those that only affect registers. Thus Δf_c is taken as:

$$\Delta f_c = FuI + \varepsilon_1 PI + \varepsilon_2 VI \ with \ \varepsilon_2 << \varepsilon_1 << 1$$

where Fui is a FU influence factor, Pi a FU ports influence factor and Vi a variable influence factor. These three factors are detailed below.

a. Functional unit influence factor Fui:

Fui represents the effect of a merge of two variables on the circuit's FUs. Its value is equal to the number of FUs that become fully controllable due to the merge. A variable merge can make a FU controllable either directly (because the FU input is linked to the non controllable variable) or indirectly (because the FU in question is down–stream a C–transparent FU linked to this variable and so on). Thus, the algorithm for computing Fui for a given variable pair (v_1,v_2) is the following (v_1 is an uncontrollable variable while v_2 is controllable, and $\mathcal{F}_\mathcal{C}$ is the set of fully controllable FUs):

```
𝒞= {v1} ; Fui(v1,v2)=0;
ℱ = ∅ ; /* set of FUs made controllable */
Until no variables are added to 𝒞
            for every FU F / F∉ℱ, F∉ℱ_𝒞 , and all input ports of F are controllable or are fed by v∈𝒞
                  Fui(v1,v2)++;ℱ=ℱ∪{F} ;
                       if F is C–transparent
                            for every uncontrollable variable v' fed by F: 𝒞 =𝒞∪{v'};
```

b. Functional unit ports influence factor Pi:

While Fui represents the influence of a merge on the controllability of a functional unit as a whole, Pi represents the effect of a merge on the controllability of input ports of a FU taken separately (i.e. even when the merge does not make the FU in question fully controllable). Pi is taken as: ,

$$\sum_{fu_i \in F} \frac{nb.\ of\ input\ ports\ of\ fu_i\ made\ controllable}{nb.of\ uncontrollable\ ports\ of\ fu_i}$$

where F is the set of uncontrollable FUs of which some input ports will become controllable due to the merge.

c. Variable influence factor Vi:

This term takes into account the effect of a merge on the controllability of registers regardless of the effect on the FU controllability. Vi=0 if the merge is between two controllable variables or two uncontrollable ones. Otherwise, the value of Vi, given in table 3, depends on the controllability classes of the merging variables, with a maximum value for merges that influence the controllability of a large number of variables and reduce sequential depth. The controllability classification of variables, issued from the testability analysis, is the following (the same classification can be obtained for any other entity):

$\mathcal{C}1$: variables controllable directly from primary input ports without FUs in the C–path.

$\mathcal{C}2$: variables $v \notin \mathcal{C}1$ and $\exists$ C–path to v with FUs in the C–path.

$\mathcal{C}3$: variables only fed by non C–transparent FUs and thus de facto uncontrollable.

$\mathcal{C}4$: variables $v \notin \mathcal{C}1 \cup \mathcal{C}2 \cup \mathcal{C}3$ and $\exists$ C–path from $v' \in \mathcal{C}3$ to v. These variables would be controllable through FUs if the $\mathcal{C}3$ variables became controllable.

$\mathcal{C}5$: $\mathcal{V} - \mathcal{C}1 \cup \mathcal{C}2 \cup \mathcal{C}3 \cup \mathcal{C}4$ = uncontrollable variables, they only belong to structural loops. $\mathcal{C}5_1$ is a subset of $\mathcal{C}5$ such that all structural loops are broken if all $v \in \mathcal{C}5_1$ became controllable ($\mathcal{C}5_2 = \mathcal{C}5 - \mathcal{C}5_1$).

Merging a non–controllable variable with a $\mathcal{C}1$ variable is preferred to that with a $\mathcal{C}2$ variable due to the sequential depth reduction resulting from such a merge. Such a merge makes a previously non–controllable point in the circuit directly accessible from a test point (e.g. a primary input). In the same way, merging a controllable variable with a $\mathcal{C}3$ or a $\mathcal{C}5_1$ variable is encouraged more than with a $\mathcal{C}4$ (respectively $\mathcal{C}5_2$) variable because if all $\mathcal{C}3$ (resp. $\mathcal{C}5_1$) variables are made controllable, then all $\mathcal{C}4$ (resp. $\mathcal{C}5_2$) variables would become controllable too.

$\mathcal{C}1$	$\mathcal{C}2$	$\mathcal{C}3$	$\mathcal{C}4$	$\mathcal{C}5_1$	$\mathcal{C}5_2$	
0	0	10	5	10	0	$\mathcal{C}1$
	0	8	3	8	0	$\mathcal{C}2$
		0	0	0	0	$\mathcal{C}3$
			0	0	0	$\mathcal{C}4$
				0	0	$\mathcal{C}5_1$

Table 3 : Variable influence factor (Vi) between register compatible variables of different classes.

V.2 Examples and Results

To show the effectiveness of our method, we applied it to several benchmark examples. Table 4 gives the characteristics of these circuits: ex1 is Tseng's example borrowed from [20]. ex2 is the differential equation example (c.f. IV) from [18]. ex3 is the AR filter borrowed from [13]. ex4 is the elliptical filter borrowed from [5]. In the last three examples, multiplication coefficients are constants and are assumed to be stored in ROMs. Thus, with regard to the C–path and O–path definitions, the multipliers can not belong to transparency paths.

circuits	# FUs	# FU C–transp.	#FU O–transp.	# var.	var. controllability	var. observability
ex1 [20]	4	2	2	11	5 c 6 nc	3 o 8 no
ex2 [18]	4	3	3	12	10 c 2 nc	6 o 6 no
ex3 [13]	5	1	4	32	4 c 28 nc	2 o 30 no
ex4 [5]	5	3	3	35	2 c 33 nc	1 o 34 no

Table 4: characteristics of the benchmarks

The number of supposed C–transparent and O–transparent functional units are given in columns 3 and 4. Column 5 gives the total number of variables of the behavioral description, while the number of controllable/uncontrollable and observable/ unobservable variables are given in columns 6 and 7. We only detail the register allocation process for one of these benchmarks: ex2 (differential equation). Fig. 4 shows the register compatibility graph of ex2. Table 5 gives the results of register allocation for ex2, once without testability considerations and once taking into account the testability of the design.

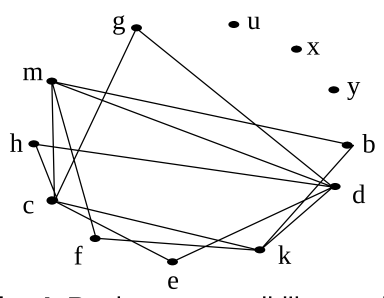

Fig. 4: Register compatibility graph of ex2

Without testability				With testability			
variables	Reg.	cont.	obs.	variables	Reg.	cont.	obs.
u	Reg1	c	o	u	Reg1	c	o
x	Reg2	c	o	x	Reg2	c	o
y	Reg3	c	o	y	Reg3	c	o
b,k	Reg4	c	o	b,m	Reg4	c	o
d,g	Reg5	c	o	d,h	Reg5	c	o
e,c	Reg6	c	o	k,f	Reg6	c	o
f,m	Reg7	c	o	e,c	Reg7	c	o
h	Reg8	nc	no	g	Reg8	c	o

Table 5: Testability results for register allocation without testability for ex2.

Table 6 gives the results of register allocation (with and without testability constraints) for the four benchmarks in terms of area cost, and testability problems. The results given below show that considerable testability improvement has been achieved with no overhead. For ex3 and ex4, register allocation for testability even resulted in fewer registers. However, certain testability problems remain unsolved. For example, for circuits ex3 and ex4, the non–controllability of certain FUs is due to the fact that one of the operands of each of these FUs is always a constant (multiplication coefficient). In such a case, register allocation cannot intervene directly on the uncontrollable FU port, but it still plays an important role concerning the testability of down–stream entities.

Remark : generally and conversely to these examples, it may happen that the number of registers differs from the initial solution (more or less registers) since clique partitioning is an NP–complete problem. So, any heuristic may not converge toward the best solution.

circuits	without testability		with testability	
	Area cost	Testability pbs	Area cost	Testability pbs
ex1 [20]	5 registers	1 reg. nc	5 reg.	None
ex2 [18]	8 registers	1 reg nc., 1 reg no	8 reg.	None
ex3 [13]	12 registers	8 reg nc, 5 FU nc 5 reg no, 1 FU no	11 reg.	2 reg nc, 3 FU nc
ex4 [5]	14 registers	1 reg. nc, 1 FU nc 13 reg. no, 3FU no	13 reg.	1 FU nc

Table 6: results of register allocation of benchmarks

During register allocation, we have used controlability and observability gains to weigh the graph edges in the compatibility graph. This method allows to improve testability in the limit of the merging possibilities with controllable/observable variables. Our aim being the generation of easily testable designs, all remaining testability problems (if any) have to be solved in the next step of the synthesis flow, that is during interconnection generation. In the next section we propose an interconnection generation methodology that takes into account testability constraints and results in an easily testable architecture with a minimal number of additional connections.

VI Interconnect network generation for accessibility enhancement

At this stage of the High Level Test Synthesis process, the data structure is composed of a set of physical modules (FUs, registers,...) and a set of data transfers between these modules. In order to solve the possible remaining testability problems, the next step consists in generating automatically an interconnect network such that modules' input ports are controllable from primary inputs and modules' output ports are observable through primary outputs.

Generally, two models of interconnect networks are used: either a bus based model or a mux based model [21]. The first one is a generalization of the second one since a bus can be viewed as a mux/dmux device. In the following, a bus based model is assumed in which modules communicate through buses. For instance, a data transfer A $\rightarrow$ B is of the form:

A –> connection –> BUS –> connection –> B.

VI.1 Basic interconnect generation algorithm

In the Mach system, the interconnect generation process is composed of two main tasks described below.

1. A preprocessing phase consists in:

i. constructing a transfer matrix M in which each data transfer between a resource output port and a resource input port is identified by a connection variable x_i. Each row in M contains connection variables corresponding to transfers from a module's output port, each column contains connection variables corresponding to transfers to a module's input port (see example in Fig. 3.a). For instance, in this example, connection variables x_1 and x_2 are respectively associated to transfers from Alu1(O1) to R1(I1) and from R2(O1) to Alu2(I2)

ii. setting up a set of ‘‘parallelism'' constraints between these connection variables. A ‘‘parallelism'' constraint exists between x_i and x_j if data transfers identified by these variables are executed during the same control step.

2. Then, a solution is sought minimizing the number of connections between entities for a given number of buses. Let B be the chosen number of buses, and let $\mathfrak{B}$: x_i –> $\mathfrak{B}(x_i) \in \{1,\ldots ,B\}$ be the searched mapping function. The algorithm attempts to find $\mathfrak{B}(x_i)$ for every x_i such that:

$$\sum_{l} Card(\{\mathfrak{B}(x_{l1}), \mathfrak{B}(x_{l2}), \ldots, \mathfrak{B}(x_{li})\}) + \sum_{c} Card(\{\mathfrak{B}(x_{c1}), \mathfrak{B}(x_{c2}), \ldots, \mathfrak{B}(x_{ci})\})$$

(where $l \in$ matrix rows and c belongs to matrix columns) is minimum while respecting the parallelism constraints, which are expressed as $\mathfrak{B}(x_i) \neq \mathfrak{B}(x_j)$. For instance in Fig.3.a, if $\mathfrak{B}(x_2)=1$, the output of R2 will be connected to Bus1 as well as the second input of Alu2. It must be noticed that the cost of the solution (number of connections) increases with the number of parallelism constraints.

Controllability and observability properties are extended to connection variables using the definitions below:

1.The connection variable x_i is **controllable** (resp. observable) if there is one module's controllable output (resp. observable input) port connected to x_i.

2.The **controllable** (resp. observ.) **bitwidth** of x_i, $CB(x_i)$ (resp. $OB(x_i)$), is the number of controllable (resp. observable) bits through x_i. Let $CB(O_j)$ be the number of controllable bits on a module output port O_j, $CB(x_i) = \text{Max}\ \{CB(O_j)\ /\ O_j$ is connected to $x_i\}$.

VI.2 Principle

Controllability and observability problems are solved by imposing to some data transfers to be executed through the same bus. In the above formalism, this is done by forcing the the corresponding connection variables to be assigned to the same bus.

For instance, let us consider a data transfer of the behavioral description from a module's output port O to another module's input port I, and let us assume that I is not controllable. Let x_i be the connection variable associated to the data transfer from O to I. Let x_j be a controllable connection variable associated to another data transfer from a controllable output port O' to an input port I'. Assuming that $CB(x_j)$ is large enough to control I and no parallelism constraint exists between x_i and x_j, forcing these two data transfers to be assigned to the same bus makes I controllable through the new transfer possibility from O' to I. This is achieved by setting up the constraint $\mathfrak{B}(x_i) = \mathfrak{B}(x_j)$. Such constraints are called test constraints and are to be set before the processing step in the above algorithm. Conversely to parallelism constraints which prevent the assignment of two connection variables to the same bus, a test constraint leads to the assignment of two variables to the same bus.

The test constraint generation for controllability is illustrated by the example in Fig. 5. For convenience, we will assume that all ports have the same bitwidth. The R2's output port, R2(O1), is controllable and the Alu2's input port, Alu2(I2), is controllable through R2(O1). As a consequence, x_2 is a controllable connection variable and all input ports which will be connected to x_2 will be controllable through R2(O1) and x_2. In the same way, R1(I1) is not controllable since it is only connected to not controllable outputs.

If there are no parallelism constraints between x_1 and x_2, a test constraint $\mathfrak{B}(x_1)=\mathfrak{B}(x_2)$ forces the two transfers, R2(O1) → Alu2(I2) and Alu1(O1) → R1(I1), to be executed through the same bus. A transfer possibility is created between R2(O1) and R1(I1) which becomes controllable through R2(O1). In the transfer matrix, the test constraint $\mathfrak{B}(x_1)=\mathfrak{B}(x_2)$ is applied replacing x_2 by x_1 in M. The interconnect generation process working on the new transfer matrix will assign the same bus (B1) to the two preceding transfers (Fig. 5.b).

$$\begin{array}{ll|ccccc|}
 & & \text{R1(I1)} & \text{Alu2(I2)} & \text{R3(I1)} & \text{Alu4(I1)} & \dots \\
 & & \text{nc} & \text{c} & \text{c} & \text{nc} & \\
\text{Alu1(O1)} & \text{nc} & x_1 & & & & \\
\text{R2(O1)} & \text{c} & & x_2 & & & \\
\text{Alu2(O1)} & \text{c} & & & x_3 & & \\
\text{Alu3(O1)} & \text{nc} & x_4 & & & & \\
\text{R1(O1)} & \text{nc} & & & & x_5 & \\
\vdots & & & & & &
\end{array}$$

$$\text{M}$$

$$\Big\downarrow \text{ Test constraint } (x_1=x_2=x_{100})$$

$$\text{new M}$$

$$\begin{array}{ll|ccccc|}
 & & \text{R1(I1)} & \text{Alu2(I2)} & \text{R3(I1)} & \text{Alu4(I1)} & \dots \\
 & & \text{nc} & \text{c} & \text{c} & \text{nc} & \\
\text{Alu1(O1)} & \text{nc} & x_1 & & & & \\
\text{R2(O1)} & \text{c} & & x_1 & & & \\
\text{Alu2(O1)} & \text{c} & & & x_3 & & \\
\text{Alu3(O1)} & \text{nc} & x_4 & & & & \\
\text{R1(O1)} & \text{nc} & & & & x_5 & \\
\vdots & & & & & &
\end{array}$$

Fig. 5.a:Transfer Matrix modification

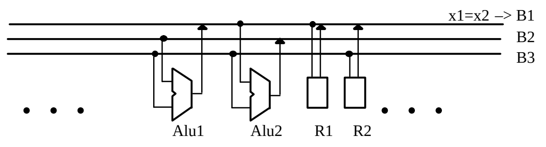

Fig. 5.b: Impact on interconnect generation

The interconnection unit generation with testability improvement algorithm is the following :

```
connect_generation_including_test_constraints() {
create_transfer_matrix;
create_parallelism_constraints;
if (test improvement) {
    test_analysis(behavioral transfers);
    if (hard to control points)
        controllability_constraints();
    test_analysis(behavioral transfers + added transfers)
    if (hard to observe points)
        observability_constraints();
    }}
generate_connect(transfer_matrix);}.
```

Testability analysis is applied a first time taking into account transfers specified in the initial behavioral description. After the controllability constraints generation process, a second testability analysis takes into account behavioral transfers plus transfer possibilities created by this process (cf. VI.3.C).

Controllability constraints generation is detailed in the following sub–section, observability constraints are generated in a similar way.

VI.3 Controllability constraints generation

We call objectives the test problems to solve, i.e. the module input ports to be made controllable.

It must be noticed that when a test constraint $(x_i=x_j)$ is set up, all parallelism constraints for x_i become parallelism constraints for x_j and vice versa. As the solution cost increases with the number of parallelism constraints and the number of parallelism constraints increases with the number of generated test constraints, the solution cost depends on the test constraints set chosen to solve the objectives.

Generally, several test constraints can be set to solve a given controllability problem. In addition, if there are several test problems to solve we have to choose what problem to solve first. These two points are illustrated in the preceding example (cf. Fig. 5):

i. The controllability problem on R1(I1) can be solved by setting up one the test constraints $\mathfrak{B}(x_4)=\mathfrak{B}(x_2)$, $\mathfrak{B}(x_1)=\mathfrak{B}(x_3)$ or $\mathfrak{B}(x_4)=\mathfrak{B}(x_3)$.

ii. If the objective R1(I1) is solved first, then R1(O1) becomes controllable too because of the C–transparence of R1. Since there is a transfer from R1(O1) to Alu4(I1), Alu4(I1) becomes also controllable. Conversely, solving the objective Alu4(I1) first, does not solve the objective R1(I1). In the remaining, we say that the controllability of R1(I1) *implies* the controllability of ALU4(I2).

As explained above, we want to generate as few as possible additional parallelism constraints in order to obtain a solution as close as possible to the optimal solution which would be generated without testability constraint. Like the interconnect generation process itself, the identification of a minimal set of test constraints to is an NP–complete problem. We use two heuristics to choose i) what objective has to be solved first and ii) what test constraint has to be applied to solve this objective.

A. Objective priority:

When several objectives are identified by testability analysis, a priority degree (D) is used to rank these objectives. D of an objective is proportional to the number of modules whose controllability depends on that of the objective and takes into account the controllability class of the objective. D takes also into account the nature of the objective: the priority degree for a FU should be higher than for a register (as testability bottlenecks usually concern large modules). Finally, D should be higher for an objective that makes a FU fully controllable rather than partially controllable. Thus, $D = a \times df + b \times cf + c \times clf + d \times if$ where a,b,c,d are balancing factors.

The term df reflects the relative difficulty for testing FUs, RAMs and registers. In the first prototype coefficients are the following: df=1000 if the objective is a FU, df=500 if it is a RAM, df=0 if it is a register.

The second term clf is the ratio of controllable input ports of the objective versus the total number of its input ports. This term favours the control of all input ports of a same module rather than partial control of many modules.

Based on the results of the controlability classification issued from testability analysis, the third term favours input ports where controllability cannot be indirectly set up by the controlability of other ones: clf=10 if controllability class of objective is $\mathcal{C}3$ or $\mathcal{C}5_1$, clf=0 if controllability class of objective is $\mathcal{C}4$ or $\mathcal{C}5_2$.

The last term if is the number of output ports made controllable by objective resolution, it allows to take into account controllability 'implications' in the testability improvement process.

B. Choice of the test constraint:

As presented in the previous example, an objective can be solved in several ways that is to say using different controllability constraints. In the transfer matrix, if x_i is a connection variable in the column of the objective, any possible association of x_i with any controllable connection variable x_j is a control constraint allowing to solve the objective (it is clear that all associations between all x_i and all x_j are not possible due to the bitwidth to control and to parallelism constraints).
A cost function is set up in order to choose the most suitable test constraint $\mathcal{B}(x_i)=\mathcal{B}(x_j)$ among all, allowing to resolve an objective. As the best test constraint is one that minimizes the number of additional parallelism constraints, the cost function naturally represents the difference between the number of parallelism constraints before and after the application of the test constraint.

C. Test constraint algorithm:

The outline of the algorithm for test constraints generation is:
– Test constraints are generated according to the ordering given by D and C and the connection matrix is modified accordingly.
– If not all test problems are solved and if the user allows extra connections to be added, the algorithm proposes to add transfers from controllable nodes to the uncontrollable ones. These transfers are considered in the same way as the other transfers in the interconnect generation process.

It must be noticed that the preprocessing phase of test constraints generation for controllability and observability is negligible in comparison with the global CPU time of interconnection unit generation, the complexity of which is $O(|X|^{NB})$ where X is the set of connection variables and NB is the number of busses. Furthermore, since testability constraints reduce the number of connection variables in the transfer matrix, test improvement method contributes to speed up the processing phase.

VI.4 Example

To illustrate this method, we ran it on the example circuit ex5 whose behavioral description is presented by its scheduled and partly allocated data flow graph in Fig. 6. The C– and O– transparencies of the FUs are given in table 7 (F1 and F2 are O–transparent only for one of their inputs I2). Table 8 gives the controllability and observability characteristics of the circuit's modules after register allocation. Due to lack of merging possibilities with observable variables, all observability problems cannot be resolve by a register allocation process, and R1 and R3 stay unobservable.

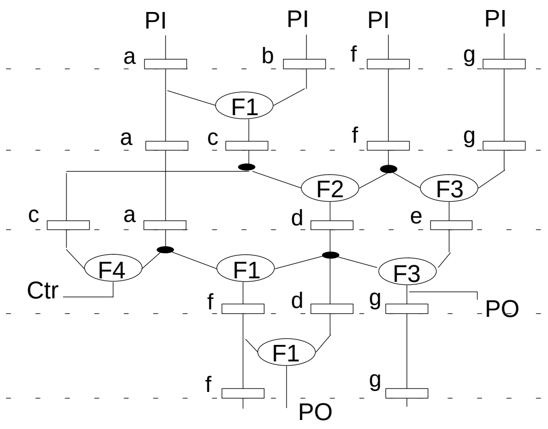

Fig. 6 Scheduled and partly allocated data flow graph of ex6

	F1	F2	F3	F4
C–transparent	yes	yes	yes	yes
O–transparent	no(I1), yes(I2)	no(I1), yes(I2)	yes	no

Table 7: C– and O–transparency properties of FUs used in ex6

Reg.	controllability	observability	FUs	controllability	observability
R1 (a)	c	no	F1	c	o
R2 (b,d)	c	o	F2	c	o
R3 (c)	c	no	F3	c	o
R4 (e,g)	c	o	F4	c	–
R5 (f)	c	o	–	–	–

Table 8: Testability results on ex6 after register allocation

Two interconnect networks have been generated, the first one in Fig. 7.a without considering testability, the second one, Fig.7.b, using test constraints. In both cases, the chosen number of busses (6) is the lower bound allowing to implement the parallelism declared in the behavioral description and the number of generated connections is 28. In Fig. 7.a, R1 and R3 are still unobservable. These problems are solved in Figure 7.b forcing the transfers from R1 to F1(I1) and from R5 to F3(I1) to be executed through the same bus B1, and forcing the transfers from R3 to F2(I1) and from R2 to F3(I1) to be executed through the same bus B2.

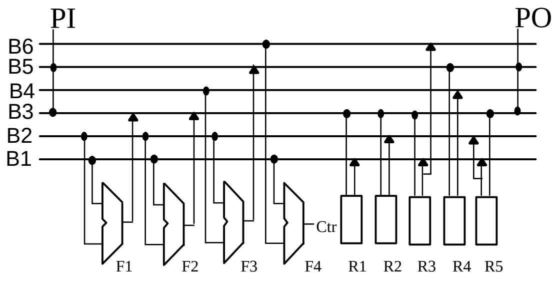

Fig 7.a Interconnect network without test constraints

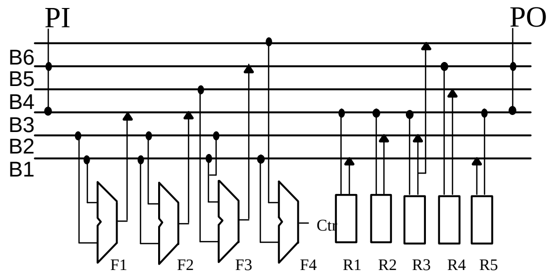

Fig. 7.b Interconnect network with test constraints

It must be noticed that for several examples, many (but not necessarily all) testability problems are solved without test constraints due to the bus based model. In fact, this model creates numerous paths between modules which are not expected by the initial behavioral description of the circuit. We expect more significant results with the mux based model of architecture.

VII Conclusion

In this paper, we presented a high level synthesis system for easy testability. A register allocation and interconnection generation schemes that take into account testability constraints are proposed. They are based on a testability analysis acting at the behavioral level. These high level synthesis tasks consider testability as one of the design parameters to be optimized in addition to area and speed. A set of testability constraints are derived from a testability analysis process and hardware sharing possibilities are used to enhance the testability of the circuit for a minimal area overhead. Register allocation is performed according to the testable/not testable characteristic of variables in order to make the largest number of registers and FUs testable. Considerable testability improvement has been obtained for several examples and for almost no overhead. Interconnection generation is performed, taking into account testability problems that have not been settled during register allocation, to ensure justification and propagation paths for all entities in the circuit. Several benchmark circuits were synthesized using this new method and results show that testability improvement is obtained for low area overhead.

References

[1] M. S. Abadir, M.A. Breuer, "A knowledge–based System for Designing Testable VLSI chips", IEEE Design&Test, pp 56–68, Aug. 1985.

[2] L. Avra, "Allocation and assignment in high–level synthesis for self–testable data paths", proc. ITC 91, pp 463–472.

[3] S. Bhatia, N.K. Jha, "Genesis: A Behavioral Synthesis for Hierarchical Testability", proc. European Design and Test Conference, pp: 272–276, 1994.

[4] C. H. Chen, C. Wu, D. G. Saab, "Accessibility Analysis on Data Flow Graph : An Approach to Design for Testability", ICCD, pp 463–466, 1991.

[5] P. Dewilde, E. Deprettere, R. Nouta , "Parallel and pipelined VLSI implementation of signal processing algorithms", In VLSI and Modern Signal Processing. S.Y.Kung, H.J.Whitehouse, T.Kailath Editors. Prentice Hall. pp: 257–260.

[6] S. Dey , M. Potkonjak, "Non–Scan Design for testability of RT–Level Data Paths", Proc. of ICCAD 94, pp: 640–645.

[7] D. Dupont, B. Rouzeyre, and G. Sagnes, "Component Selection, Scheduling, and Control Schemes for High Level Synthesis", proc. European Design and Test Conference, pp 580–585, 1994.

[8] S. Freeman, "Test Generation for Data–Path Logic : The F–path method", IEEE Journal of Solid State circuits, vol. 23, n 2, pp 421–427, April 1988.

[9] C.H. Gebotys, M.I. Elmasry , "Integration of Algorithmic VLSI Synthesis with Testability Incorporation", IEEE Journal of Solid–State Circuits, vol. 24, no. 2, April 1989.

[10] C.H. Gebotys, M.I. Elmasry , "Normal and Test Mode Design Synthesis",report UW/ICR 89–01, Institute for Computer Research, University of Waterloo,Waterloo, Ontario, Canada, 1989.

[11] D.Hammad, M.L.Flottes, B.Rouzeyre, "An analysis based approach for Easy Testable Data–Paths Synthesis", report LIRMM 1994, Université de Montpellier, France 1994.

[12] S. Hellebrand, H.J. Wunderlich, "Synthesis of Self–Testable Controllers", proc. European Design and Test Conference, pp: 580–585, 1994.

[13] R. Jain, K. Kucukcaker, M.J. Mliner, and A.C. Parker, "Experience with ADAM synthesis system", proc. DAC, pp:56–61, 1989.

[14] T–C. Lee, W. H. Wolf, N. K. Jha, J. M. Acken , "Behavorial Synthesis for Easy Testability in Data Path Allocation", proc. ICCD, pp 29–32, 1992.

[15] T–C. Lee, N. K. Jha, W. H. Wolf , "Conditional resource sharing method for behavioral synthesis of highly testable data paths", proc. ITC 93, pp :744–753.

[16] M. C McFarland, A.C. Parker, R. Camposano, "Tutorial on High–Level Synthesis", proc. DAC, pp 330–336, 1988.

[17] C. A. Papachristou, S. Chiu, H. Harmanani, "A Data–Path Synthesis Method for Self–Testable Designs", proc. DAC, pp 378–384, 1991.

[18] P.G. Paulin, J.P. Knight "Scheduling and binding algorithm for high level synthesis", proc. DAC, 1989, pp. 1–6.

[19] B. Rouzeyre, G.Sagnes, "A new method for the minimization of memory related area", EURO–ASIC 91, Paris, 28–31 Mai 1991.

[20] C.–J. Tseng and D. Siewiorek, "Automated Synthesis of Data Paths in Digital Systems", IEEE Transactions on CAD, vol.5, No. 3, July 1986.

[21] C.H. Tseng and D.P.Siewiorek, "The modeling and synthesis of bus vsystems", proc. DAC, June 1981, pp: 471–478.