Analyzing testability from behavioral to RT level

M.L. Flottes, R. Pires, B. Rouzeyre

Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier,
Université de Montpellier 2, UMR CNRS 5506, 161 rue Ada, 34392 Montpellier Cedex 5, FRANCE

Abstract

In this paper, we present a method for analyzing the testability of a circuit during high level synthesis. The testability analysis returns values that represent the relative difficulty for computing test data, whatever the level of description of a circuit is (from the behavioral level –initial specification– down to the Register Transfer Level –high level synthesis output–). Experiments show the good correlation of the so–obtained testability measures with gate–level testability measures (e.g. Scoap). The proposed measures are used to guide high level synthesis towards the generation of easily SATPG testable datapaths.

I Introduction

The testability analysis method proposed here takes place in a High Level Synthesis (HLS) flow, where typically, an input behavioral specification is synthesized in an output structural representation at Register Transfer Level (RTL). Many RTL designs can be generally synthesized from a given behavioral description. They may achieve the same goals for several characteristics (area, performances, ...) but testability. The designs are composed of a datapath and of a controller. In this paper, we address the testability of the datapath assuming that the controller can be modified to be self–testable (e.g.[1]), and that a test mode allows to test the datapath regardless to the operations defined by the system mode.

All along the synthesis process, the internal representation of the design is a Data Flow Graph (DFG). Initially the nodes in the DFG are variables and operations, then these behavioral elements are progressively mapped to structural elements (registers, operators) as the HLS proceeds. An oriented edge from node A to node B, $A \rightarrow B$, represents the elementary data transfer which has to be performed between the datapath components A and B. A set of data transfers starting from A and ending at B, $A \rightarrow ... \rightarrow B$, is called a path from A to B. The DFG depicts the elementary data transfers between datapath components and the paths activated during the system mode.

The goal of the method presented here is to early identify and predict testability problems in the RTL structure to come, using information issued from its DFG representation, actually by scanning elementary data transfers.

The paper is organized as follows. Related works are discussed in the following section. Proposed testability constraints and measures are detailed in section III. The testability analysis process is presented in section IV. The fifth section briefly describes how testability information are used to guide the HLS process to the generation of easily SATPG testable structures (SATPG stands for Sequential Automatic Test Pattern Generation). Experimental results are presented in the section VI.

II Related works

The testability issues targeted by most of HLS for testability systems (e.g. [2], [3], [4]) are mainly sequential depth reduction, loops reduction, and the increase of I/O registers. Most of these works do not consider the other testability bottlenecks induced by reconvergences or module transparence properties [5]. Furthermore, these works do not rely on testability measures.

Concerning methodologies based on a testability analysis ([6], [7]), the testability of the nodes is established by analyzing only the paths activated during the system mode. For example in [6], a variable is of type complete controllable if there exists a sequence of executable paths in the DFG such that after the execution of these paths, the content of that variable can take any possible value by adjusting the input values. These testability results are used to modify the input behavioral specification in order to increase accessibility. In the Genesis system [7], limited transparency properties of operators (neutral element is used as one of the operands : 0 for additions, 1 for multiplications ....) are used to check the existence of justification and propagation paths for the operations in the DFG. It must be noticed that, in both approaches, the testability of the nodes is simply two–valued (testable or not testable).

III Basics

The method presented here gets rid off the above limitations by considering all potential paths allowing to reach any internal node in a datapath from I/O. The testability analysis process determines the probability for justifying and propagating any test data to internal nodes. These probabilities are computed given 1/ the transparency coefficients of operations, and 2/ the set of elementary data transfers performed between datapath components. Transparency coefficients are computed for all kinds of functional units and are not based on particular algebraic properties (e.g. the existence

ED&TC '97 on CD-ROM

of a neutral element). Thus, the application domain of this method is not restricted to some special cases of circuits. Furthermore, the testability analysis returns corresponding justification/propagation paths for controllable/observable nodes. Those paths are built up from the elementary data transfers to be performed during the system mode flow regardless of their sequencing.

Note that the values of test patterns and of test responses can not be derived at behavioral level for actual physical faults. As a matter of fact, this would mean that, at least for justification, the reverse function of any operation should be known to set the appropriate values on its inputs for a given test patterns required on its output. According to this, we define, for a n–bitwidth node N, the following testability measures:

– Controllability measure $C(N) = y / 2^n$, where y is the number of any patterns (test pattern or not) that can be propagated to this node from the primary inputs.

We liken this proportion to the probability of propagating the necessary test patterns to the node.

– Observability measure $O(N) = y / (2^{n-1} * (2^n - 1))$, where y is the number of pairs of different responses that can be propagated and differentiated at the primary outputs regardless of their values, and $(2^{n-1} * (2^n - 1))$ is the total number of possible pairs.

Here too, we consider O(N) as being the probability to differentiate fault–free and faulty responses of this node.

These testability measures are ranging in [0..1]. With regard to these definitions, the highest those measures are, the highest the probability of deriving test patterns at gate level is. Since it is impossible to derive directly these values, approximations of these metrics are established by taking into account both transparency and data transfer information that can be derived from the DFG. The approximation computation is presented in chapter IV.

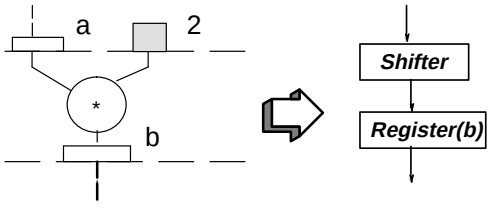
Even in the absence of structural model, some behavioral information can be used to predict some future structural implementations which may result in testability bottlenecks. The testability bottlenecks are related either to the impossibility or to the difficulty of propagating test data in the circuit. The former ones are discussed below:

1/ Transparency: Due to their functional characteristic, some of circuit's modules, namely functional units (F.U.s), are not favorable to test data propagation. For example in the Fig.1.a, if the fragment of the DFG represented on left side contains the only multiplication of the specification, then only even values can be obtained at the output of the operator implementing this operation. Furthermore, if there is no other operation to ‘‘produces” the variable b, then, neither the register implementing b nor downstream modules can be fully tested.

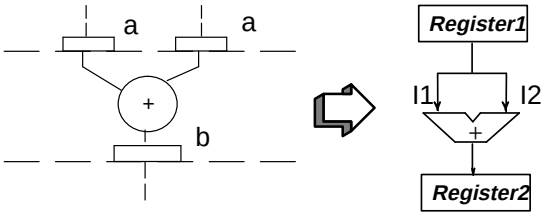
2/ Reconvergence: A module presents a controlability problem if its input ports cannot be set independently to any value. For instance, assuming that the operation depicted in Fig.1.b. (left side) is the only addition in the specification, since both inputs of this addition come from the same variable a, there is a divergence/reconvergence problem between register1 and register2 respectively used to store a and b in the corresponding structural implementation (as depicted on the right side of Fig.1.b). Consequently, controlability problems arise for register2 and downstream modules.

3/ Loops: Two kinds of loops can be distinguished, namely ‘‘strongly closed loops” and other loops. Strongly closed loops are such that all registers in the loop can only be reached through the registers involved in the loop. Thus full control from primary input ports PIs is not possible (see Fig.2.a). It must be noticed that this kind of loop appears in a RTL structure only when there are loops in the behavioral specification. Conversely, a loop is not strongly closed if it exists some paths to justify values on the registers involved in the loop (see Fig.2.b). Eventual test problems in such loops are related to the search of these justification paths and not to the absence of these paths as in strongly closed loop.

About later testability bottlenecks (general loops, sequential depth), it can noted 1/ that present commercial SATPGs are most of the time able to find test paths if they exist (testability metrics can be used in order to prune the search space), and 2/ that the sequential depth, which is well known to be a second order SATPG slowing down factor, is not large in datapaths. These points are nor further discussed in this paper.

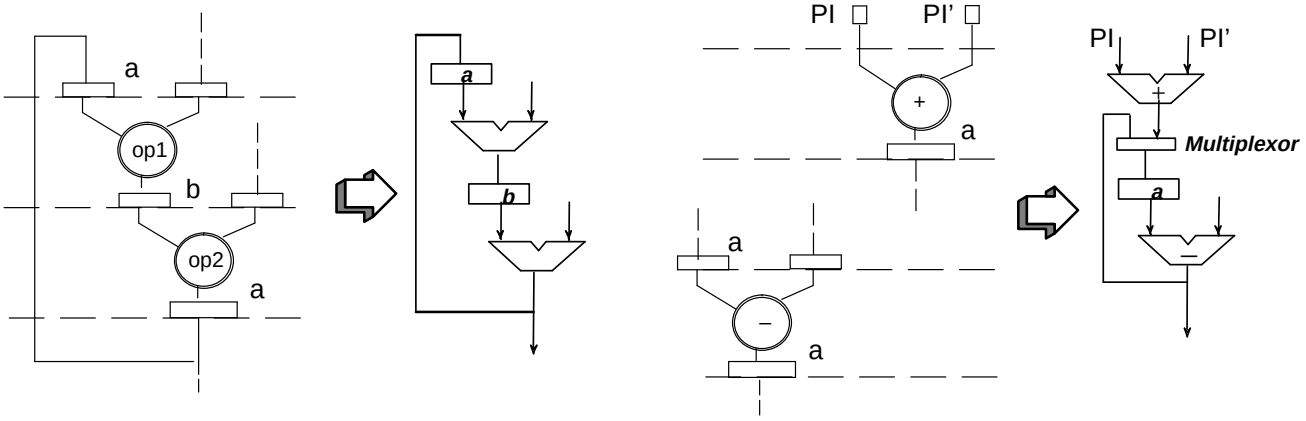


(a)Transparency: Test pattern 01 cannot be justified on the register input



(b) Reconvergence: Test pattern {01,10} cannot be justified on the adder inputs {I1,I2}

Fig.1 Testability Bottlenecks



(a)Strongly closed loop: value sets into Ra depends of the preceding value in Ra.

(b) Other loop: R1 can be set to any value from primary inputs.

Fig.2 Testability Bottlenecks

IV Testability analysis

IV.1 Transparency

In order to derive the testability measures, we use the two following transparency coefficients for F.U.s (or operations):

Considering a m–bitwidth output of an isolated F.U., its controlability coefficient Tc is the ratio of different values that can be set on the F.U. output over 2^{m}. In order to take into account possible correlation between input ports (e.g. Fig.1(b)), $Tc = \frac{(C2–C1)\ .\ p}{n} + C1$, where n is the bitwidth of the inputs ports, p is the number of common bits between the input ports, C1 (resp. C2) is the proportion of values that can be obtained on the F.U. output given that its inputs ports are not connected to each other: p=0 (resp. are connected: p=n). For instance, for an adder, C1 = 1 and C2 = 0.5, for a left–side shifter, C1 = C2 = 0.5.

Observability coefficient To is the proportion of pairs of input values that can be distinguished on the isolated F.U. output. For instance, for a left–side shifter (n input bits, n output bits, with l.s.b. set to 0), $To = (2^{n}–2) / (2^{n}–1)$.

The coefficients C1, C2 and To are intrinsic to F.U.s, they are computed off–line synthesis and are included in the F.U. library as an attribute.

IV.2 Controlability

Controlability measures and justification paths are computed before observability measures and propagation paths since some control values need to be justified for propagation of test responses.

According to the above definitions, controlability measures C(.) and justification paths Jpath(.) are derived by scanning the elementary data transfers (edges in the DFG) using the algorithm given in Fig.3.

Fig.4 illustrates this process. On the left side, a fragment of a behavior is depicted where 2 F.U.s are used to implement the 4 operations. Their coefficient C1 are respectively equal to 0.6 and 0.7. A justification path for the variable g is given on the right side. Controlability measures are shown in bold characters. This justification path is built from elementary data transfers of the system mode but it does not exist as it in the DFG.

remarks :

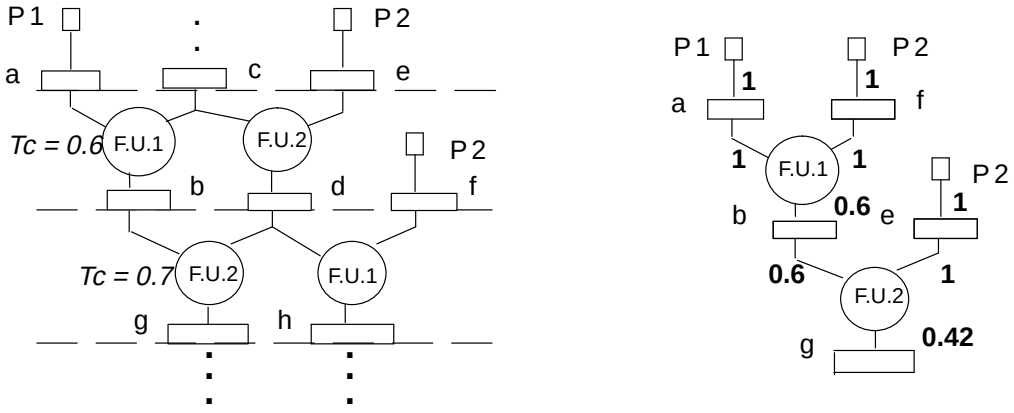
r1 This way of computing controlability may be pessimistic regarding actual test patterns. First, lets assume that for a module M input M(I), C(M(I)) = 0.5. This means that only 50% of all patterns can be propagated to this node, which is a quite bad testability value while those 50% may contain the all set of actual test patterns for testing M. Secondly, lets assume that two justifications paths P1 and P2 lead to two different C(M(I)) values, e.g. $C_{P1}(M(I))=0.5$ and $C_{P2}(M(I))=0.3$, the controllability value of M(I) is kept to 0.5 while its actual value should range in [0.5, 0.8].

r2 All variables x (or registers) belonging to strongly closed loops have a controllability measure C(x) = 0.

r3 If the operations in the DFG are not yet mapped to F.U.s, the corresponding transparency coefficients are those of the F.U.s with the highest coefficient Tc which can implement the operations. For instance, if an ALU is able to perform addition and shift, the controlability coefficient associated to the shift operation is the one of the ALU rather than the one of a shifter.

Initially, C (x) = 1 for all primary inputs nodes x. C(output(x)) = $1/2^n$ for all n–bitwidth constant nodes. C(output(x)) = $m/2^n$ for all m–words n–bitwidth ROMs. C(input(x)) = C(output(x)) =0 and Jpath(x) = {} for inputs and outputs of other nodes.
While changes occur on any C(.)
  for each elementary data transfer of the DFG
    Lets consider a data transfer (Dt) from the output of a module "a" to one input of module "b":
    if C(output(a))>C(input(b))
      then C(input(b)) = C(output (a)) and Jpath(input(b)) = Jpath(output(a)) ∪ Dt.
      else C(input(b)) and its current justification path are unchanged.
    if the module b is not a F.U. . /* variable, register, multiplexer, bus */
      then C(output(b)) = Max (C($input_i$(b)) and Jpath(output(b)) = Jpath($input_{i0}$(b)) where i0 is the input leading to the maximum.
      else /* b is a F.U. */,
        Tc(b) is set according to possible correlation between module b inputs ($input_1$(b), $input_2$(b), ..., $input_n$(b)).
        temp = Tc(b) * Π C($input_i$(b)).
        if (temp > C(output(b)) **then** C(output(b)) = temp ;
        Jpath(output(b)) = ∪ Jpath($input_i$(b))
  endfor
endwhile

Fig.3 : Controllability Measure Algorithm



DFG of the system mode

Justification path for variable g

Fig.4 Controlability measure computation and justification path

IV.3 Observability

As mentioned before, propagating test responses from the output of the module under test to the primary outputs involves justifying some values on side inputs on the propagation path. In this paper, we refer to the justification paths of the side inputs as lateral justification paths. Sufficient conditions to avoid fault masking problems along propagation path is that

1/ the lateral justification paths do not contain the module under test;

2/ the propagation path does not contain the module.

Due to condition 1/ above, it is necessary to check the existence of lateral justification paths not containing module M, where M is the module under test.. In the following, $C_{\overline{M}}$(M'(I)) denotes the controlability measure of a module input M'(I) and $JPath_{\overline{M}}$(M'(I)) is the corresponding justification for M'(I) not passing though M. $C_{\overline{M}}$(.) are computed thanks to the preceding algorithm (Fig.3) slightly modified : elementary data transfers with M as an origin or destination are not considered

The algorithm for establishing observability measures O(.) and the propagation paths Ppath(.) of each module output M is given in Fig.5.

Remark : For the same reasons as explained for controlability, this way of computing observability measure O(.) may be pessimistic.

Fig.6 and Fig.7 illustrate this process. In Fig.6 is given a DFG in which operations are mapped onto F.U.s. Ppath and lateral Jpaths for F.U.3 are respectively depicted by normal and thick lines in Fig.7.a/. The elementary data transfers used to build them are depicted in thick lines in Fig.6.

for every module M :
Initially, O(x) = 1 for all primary outputs and 0 for all inputs and outputs of other nodes.
Until no change occurs on any O(.) :
for each data transfer /* let Dt be a data transfer from the output of module a to the input of module b */
if O(input(b)) > O(output(a)) **then** O(output(a)) = O(input(b)) and Ppath(output(a)) = Ppath(input(b)) ∪ Dt.
if a is neither a F.U. nor the module M,
then
for each input of a:
if O(output(a)) > O(input(a)) **then** O(input(a)) = O(output(a)) and Ppath(input(a)) = Ppath(output(a))
endfor
else
if a ≠ M /* a is a F.U. */
then
for each input i = i_1, ..., i_n of a
temp = To(b) * O(output(a)) * Π $C_{\overline{M}}(j)$ (with j ≠ i)
if temp > O(i) **then** O(i) = temp and Ppath(input(a)) = Ppath(output(a))
endfor
else /* a = M : do nothing , see condition 2 in section IV.3 */
endfor
endwhile
– O(output(M)) and its Ppath(M) are stored.
endfor

Fig.5 Observability measure algorithm

It can be noted in this example that no propagation path can be set for the output of F.U.5 since the lateral Jpath of its potential Ppath would contain F.U.5. This can be seen on the implementation of the circuit, depicted in Fig.7.b (where there is a one–to–one mapping of variables into registers): Ppath of F.U.5 contains F.U.7 and both F.U.7 inputs depend on F.U.5. Thus, no lateral $JPath_{\overline{F.U.5}}$(F.U.7) can be established.

V Exploitation during high level synthesis

Since this testability analysis deals with different levels of description, it can be used at any moment within the high level synthesis process and particularly during the hardware binding steps.

Hardware sharing possibilities as well as information delivered by the analysis can be used in order to improve the testability of the whole design. For instance during register allocation, it is preferable to merge variables with low testability measures together with variables with high testability measure in order to produce testable registers.

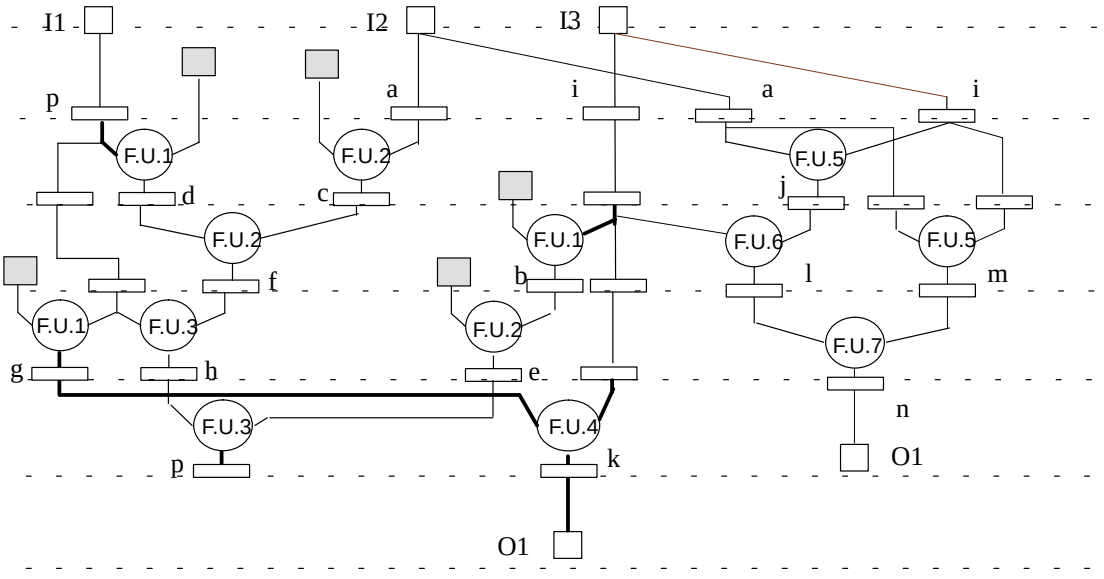
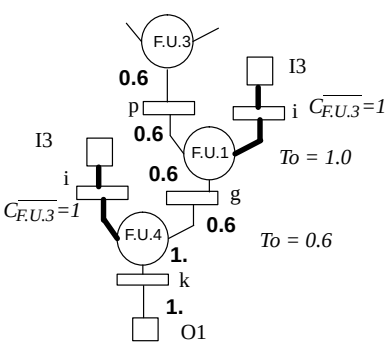
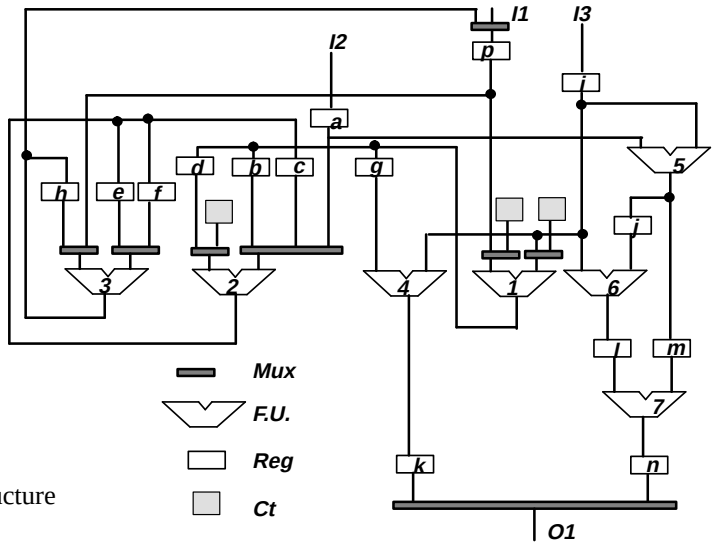


Fig.6 DFG example



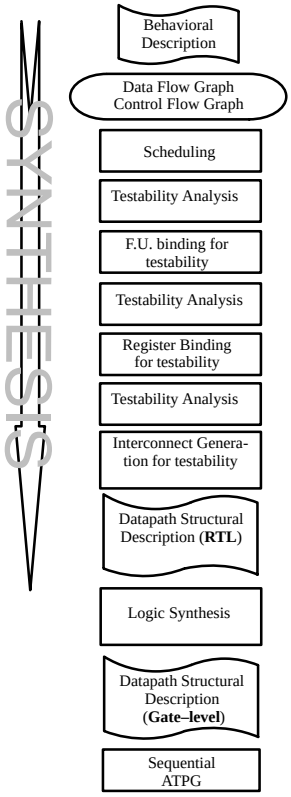
a/ Propagation path of F.U.3.



b/ Structure

Fig.7 Observability coefficient computation and propagation path

Fig.8 illustrates how testability analysis is incorporated in the high level synthesis tool ‘‘MACH” (developed at LIRMM).



Testability analysis is performed before each binding step (F.U. binding, register allocation and binding, Interconnect units generation).

Classical binding algorithms have been modified and new algorithms use testability measures to drive binding decisions for obtaining easily testable datapaths [8].

The final RTL structural descriptions can be expanded to gate–level and evaluated by an SATPG.

Fig.8 High level synthesis for easy testability

VI Results

We present here experiments conducted with four behavioral synthesis benchmarks. Tseng’s example is borrowed from [9], the differential equation example from [10], the AR filter from [11] and the elliptical filter (EW) from [12]. Unfortunately, in these HLS examples, the operations are only additions, subtractions and multiplications which are not demonstrative for our purpose (all transparency coefficients are equal to 1). For illustration purpose, we replaced some of the operations by shift operation implemented by shifters with transparency coefficients less than 1. The multipliers we used are standard two inputs multipliers. In the two filters, they are used for implementing multiplications by constants. As a consequence, and according to the assumption on controlability, their output have controllability measures much smaller than 1.

Table 1 shows the results of the application of our testability analysis method to these examples just before the register allocation step (see Fig.8). Column 2 gives the number of non–transparent F.U.s used in the design over the total number of F.U.s. Concerning testability of variables (number of which is given in column 3), results are reported in columns 4 and 5. Column 4 (resp. 5) gives the number of variables with a controllability (resp. observability) measure less than 0.8. The two last columns give the average controllability and observability of the whole designs.

Then we finished the synthesis of the designs, once using the information provided by this early testability analysis and once without paying attention to testability. We synthesized those examples with a point–to–point style of interconnections. A one level of multiplexors network has been used to connect the modules. In this model, the connection sharing possibilities are not sought to enhance the testability. Thus the testability improvement is only due to register allocation. In the following tables, ‘example’_n denotes a design obtained without regard to testability, while ‘example’_t refers to a design synthesized from the same input specification but considering testability during synthesis.

Table 2 shows the testability analysis results at RT level. Column 3 and 4 give the number of registers with controllability (resp. observability) values less than 0.8. The same holds concerning F.U.s in columns 6 and 7. Average controllability and observability of the designs are reported in columns 8 and 9. As expected, designs obtained considering testability during HLS (examples_t) present, a priori, a better testability than standard ones (examples_n).

In order to show the correlation between our analysis and gate–level testability analysis, we expanded those designs to gate–level and ran the gate–level testability analysis method of the SUNRISE test system [13]. Results are reported in Table 3. Column 3 gives the number of nodes with SCOAP testability measures between 0 and 10, column 4 gives the number of nodes with SCOAP testability measures between 10 and 50 etc. In the last column, the number of nodes with an infinite SCOAP measure, i.e. the nodes that are a priori not testable.

These results show that as predicted by our testability analysis, all example_n contain more non–testable nodes and will be more difficult to test than their corresponding example_t. Furthermore, we verified that nodes with infinite SCOAP measures match with predicted non–testable RT entities.

We applied the commercial SATPG Sunrise [13] to the generated circuits. Table 4 shows the results. For all examples, we limited the ATPG application time to 48 hours. It must be stressed that 100% efficiency (E) is always reached for example_t while only a poor efficiency is obtained in some example_n (e.g tseng_n) even with a large CPU time. Fault coverage (FC) is always higher for example_t.

Concerning synthesis results, Table 5 compares some structural characteristics. The number of cells and the area estimations were obtained using the Synopsys suite. The area increase due to the application of testability constraints were in the worst case equal to 10%. But the ewfil example results show that the area increase is not a necessary consequence of the application of testability constraints.

VII Conclusion

In this paper, we have presented a testability analysis method dealing with high level descriptions (from behavioral to RT level). It measures the probability of propagating any test data to and from every datapath element.

As illustrated on the presented examples, the results present a good correlation with traditional gate–level testability analysis.

We have shown how the testability measures provided by this method allows to drive the HLS process to the generation of easily testable circuits.

Since this method does not deal with the actual values of test patterns, its results may be pessimistic in some cases (see section IV.2 remark R1). Nevertheless, as shown on the examples, the provided measures are adequate enough for predicting testability bottlenecks. In order to improve the measure accuracy, we are presently working on establishing testability thresholds correlated to module types for deciding more accurately their testability.

While we have presented this method in the context of test improvement without DFT, it can be easily extended to the context of HLS for BIST and partial scan (BIST or scan registers being considered as additional test points). The modification of synthesis algorithms for these test methodologies is a subject for future research.

References

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

[2] T–C Lee, N. Jha, W. Wolf: ‘‘Behavioral Synthesis of Highly Testable Data Paths under the Non–Scan and Partial Scan Environments”, DAC 93, pp 292–297.

[3] M. Potkonjak, S. Dey, R. Roy: ‘‘Considering Testability at Behavioral Level : Use of transformations for partial scan cost minimization under timing and area constraints”, IEEE Trans on CAD, Vol. 14, n.5, May 1995, pp: 531–546.

[4] A. Mujumdar, R. Jain, K. Saluja :‘‘Incorporating Testability Considerations in High–Level Synthesis”, IEEE J. of Electronic Testing, pp 43–55, Feb. 1994.

[5] 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.

[6] C–H Chen, C. Wu, D. Saab : ”Accessibility analysis on data flow graph : an approcah to design for testability”. Proc. ICCD 91, pp:463–466

[7] S. Bhatia, N.K. Jha: ‘‘Genesis: A Behavioral Synthesis System for Hierarchical Testability”, Proc. ED&TC94, pp 272–276.

[8] M.L.Flottes, D. Hammad, B. Rouzeyre, ”High–Level Synthesis for Easy Testability”, ED&T95 : The European Design and Test Conference 1995. Paris 1995. pp: 198–206.

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

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

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

[12] 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.

[13] Sunrise tests system : reference manual.

design	# non–transparent F.U.s / # F.U.s	# variables	# uncontrollable variables	# unobservable variables	controlability average	observability average
arfil	1 / 3	36	32	34	0.130011	0.066973
difeq	1 / 4	8	2	4	0.771484	0.750122
ewfil	1 / 3	39	37	38	0.043999	0.045943
tseng	2 / 4	11	6	8	0.646552	0.496324

Table 1: Analysis results at behavioral level

design	# reg.	# uncontrol-lable reg.	# unobserv-able reg.	# F.U.s	# uncontrol-lable F.U.s	# unobserv-able F.U.s	average control.	average observ.
arfil_n	10	6	8	3	3	2	0.321914	0.334245
arfil_t	10	5	8	3	2	0	0.567578	0.625244
difeq_n	4	1	2	4	1	2	0.833912	0.777778
difeq_t	4	0	1	4	1	1	0.857561	0.861111
ewfil_n	13	1	12	3	1	2	0.879580	0.097806
ewfil_t	12	0	0	3	1	0	0.947505	0.903921
tseng_n	5	2	0	4	2	0	0.757576	0.907895
tseng_t	6	1	0	4	0	0	0.932432	0.931818

Table 2: Analysis results at RT level

design	# nodes	10	50	100	500	>1k	∞
arfil_n	2777	0	467	1418	540	0	352
arfil_t	3011	0	1335	914	441	0	321
difeq_n	1136	8	802	125	74	0	127
difeq_t	1161	8	831	199	79	0	44
ewfil_n	2424	0	30	1522	820	0	52
ewfil_t	2330	0	411	1314	578	0	27
tseng_n	1381	0	843	471	8	0	59
tseng_t	1525	0	1059	441	0	0	25

Table 3: SCOAP based measures (gate–level)

design	E (%)	FC (%)	ATPG time
arfil_n	85.75	74.09	48 h
arfil_t	100	89.44	34.99 h
difeq_n	100	94.94	44.91 s
difeq_t	100	96.24	34.61 s
ewfil_n	92.31	91.40	48 h
ewfil_t	100	99.71	1.50 min
tseng_n	28.52	25.69	48 h
tseng_t	100	99.24	1.89 min

Table 4: ATPG results

design	# registers	# F.U.s	# mux	# mux inputs	# cells	area	area increase (%)
arfil_n	10	3	11	32	668	964493	——
arfil_t	10	3	13	42	759	1023541	+ 6.1
difeq_n	4	4	6	14	335	399700	——
difeq_t	4	4	6	15	341	406184	+ 1.6
ewfil_n	13	3	8	38	546	804020	——
ewfil_t	12	3	9	38	536	769256	– 4.3
tseng_n	5	4	8	19	267	463873	——
tseng_t	6	4	10	22	282	512703	+ 10

Table 5: Characteristics of the designs