On Encoding and Utilization of Diagnostic Information Extracted from Design-Data for Testability Analysis Purposes

Josef Strnadel[1]

[1]*Faculty of Information Technology, Brno University of Technology, Brno, Czech Republic*
strnadel@fit.vutbr.cz

Abstract. Estimation of testability parameter of a digital circuit strongly depends on quality of input information utilized for the estimation by a testability analysis method. In the paper, it is illustrated how the information can be encoded, utilized and extracted from design-related data. For each component from a library of basic design components, the information can be stored in the library together with other data in order to be utilized for analysis of any future design based on components from the library.

Keywords. boolean function, diagnostic information, encoding, horizontal line test, injection, mapping, surjection, transparency extraction, testability analysis, truth table

I. Introduction

A lot of research efforts have been dedicated to the importance of modeling diagnostic data transfers in digital circuit data-path for purposes of more precise analysis of circuit properties from diagnostic point of view. Probably, the first (so-called *I/T-Path*) model was published in [1]. The model supposed transfer of *n*-bit diagnostic data is possible in the direction from *n*-bit port *x* to *n*-bit port *y* in circuit data-path iff one-to-one (i.e., bijective) mapping exists between *x*-data and *y*-data.

Other works tried to enhance properties of I/T-Path based model. For example, in conception referred to as *S/F-Path conception* [3], it was shown I/T-Path conception is easy to understand and implement, but it is too strict in definition of data-paths suitable for diagnostic data-transfer. I/T-Path's strict "bijective mapping" requirement leads to unneeded restriction of set of data-paths suitable for transferring diagnostic data. The strict requirement can be soften by analyzing data-path separately for transferring test vectors (responses) between ports *x* and *y*.

The idea of such a separate analysis is as follows: test vectors (responses) can be transferred from *x* to *y* iff a surjective (injective) mapping exists between *x*-data and *y*-data. Using this less-strict principle, much more data-paths can be considered suitable for transferring diagnostic data than in case of I/T-Path conception. Also, several variations of above-mentioned surjective, injective, and bijective approaches, including *ambiguity sets* [7], *transparency modes* [11], or *transparency channels* [5, 6], have been used in the area of generating so-called *hierarchical tests*. All above-mentioned approaches are often referred to as *transparency conceptions*, because they deal with modeling of situations in which data-path portion is transparent to transported diagnostic data.

II. Our Previous Work

In our previous research activities, we tried to take advantage of existing transparency principles utilized successfully for enhancement of hierarchical test-generation methods and utilize them for enhancement of *register-transfer level* (RTL) *testability analysis* (TA) process. From all transparency principles, it was S/F-Path conception that has been found as the most suitable for solving problems dealt during our previous research activities like TA [9], *design-for-testability* (DFT) [10] and synthetic benchmark-generation [8] problems solved over the class of RTL digital circuits. In [9], it was shown testability can be estimated very precisely when S/F-Path conception is utilized for modeling diagnostic data transfers through RTL data-path. In addition, it was shown there is a very close relationship between testability values got by our academic TA tool (working in linear time, which is general assumption posed on any TA algorithm [2]) and fault-coverage values got by commercial ATPG tools (generally, working in exponential time) – in average, there was 5% deviation between our testability values and fault-coverage values gained for FITTest_Bench06 benchmarks [4].

III. Motivation of Our Research

To be able to apply transparency conception by automated methods being developed at our faculty, it was necessary to develop a method for automated extraction of transparency information first (in the past, the information was being created manually, so it was very difficult to "generate" it for components with I/O bit-widths equal or wider than 4). For each component from a library of basic design components, the information can be stored in the library together with design-related data and other data like test vectors, responses etc. in order to be utilized for analysis of any future design based on components from the library – see Fig. 1 for illustration of the role transparency information plays in our DFT system.

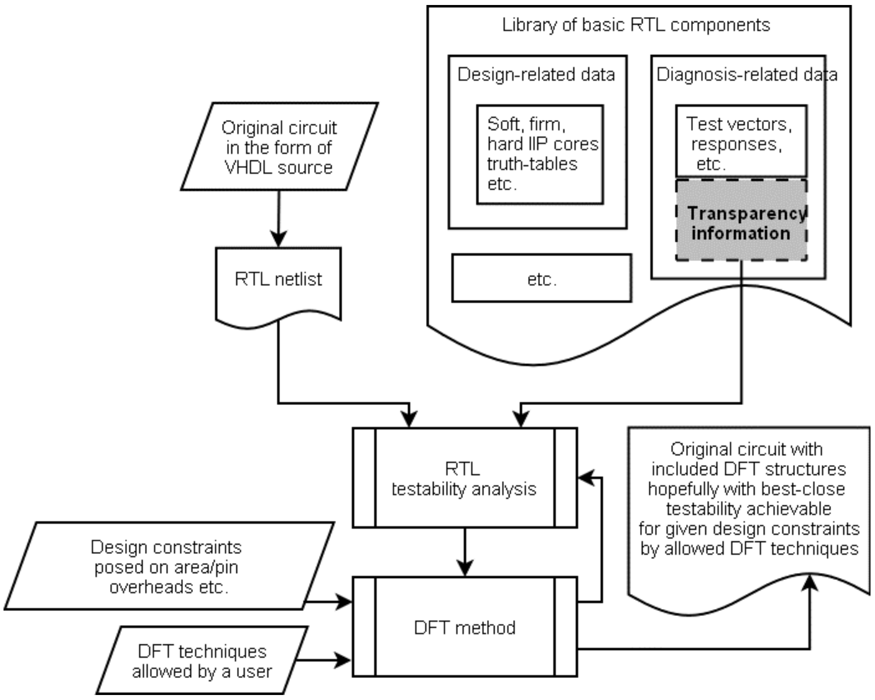
Library of basic RTL components
Design-related data
Diagnosis-related data
Original circuit in the form of VHDL source
Soft, firm, hard IIP cores truth-tables etc.
Test vectors, responses, etc.
Transparency information
RTL netlist
etc.
RTL testability analysis
Original circuit with included DFT structures hopefully with best-close testability achievable for given design constraints by allowed DFT techniques
Design constraints posed on area/pin overheads etc.
DFT method
DFT techniques allowed by a user

Fig 1 Flowchart illustrating a role of transparency information in our DFT system

IV. Principles Related to Proposed Transparency Extraction Method

In contrast to above-mentioned transparency-related papers forming complex transparency information to be utilizable for (exponential-time complexity) test-generation methods, we dealt with design of a method for extracting simple transparency information to be utilizable for (linear-time complexity) TA methods.

A. Actual Inputs and Outputs of the Extraction Method

In our actual transparency-extraction approach, it is supposed function of each basic RTL component is described by means of a truth table. For general case of k-bit output functions ($k \geq 1$), our transparency-extraction method takes a set F = $\{f_{k-1},\ldots, f_0\}$ of k n-input Boolean functions f_i: $\{0,1\}^n \rightarrow \{0,1\}$, $i = k-1,\ldots, 0$ as its input (function f_i is related to i^{th} output bit) and generates transparency information at its output (see Fig 2). Set F can be seen as a function f: $\{0, 1\}^n \rightarrow \{0, 1\}^k$, so our method takes F in this f-form representing classic truth-table (e.g., see Tab. I).

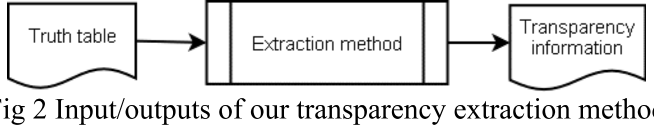

Fig 2 Input/outputs of our transparency extraction method

For illustration of principles presented in this paper simple 1-bit full-adder (FA) is utilized. Function of FA module can be described by (3-input, 2-output) truth table (presented in Tab. I), which is an input to our actual transparency extraction method.

For general function f: $\{0,1\}^n \rightarrow \{0,1\}^k$, let $\mathrm{IN_f} = \{f_{\mathrm{in}}(n\text{-}1),\ldots, f_{\mathrm{in}}(0)\}$ be the set of its n inputs and $\mathrm{OUT_f} = \{f_{\mathrm{out}}(k\text{-}1),\ldots,f_{\mathrm{out}}(0)\}$ be the set of its k outputs. For FA, $\mathrm{IN_f} = \{\mathrm{x, y, c_{in}}\}$, $\mathrm{OUT_f} = \{\mathrm{z, c_{in}}\}$. Also, let ν: $\mathrm{IN_f} \cup \mathrm{OUT_f} \rightarrow \{0,1\}$ be a mapping assigning a bit-value appearing at particular input/output $x \in \mathrm{IN_f} \cup \mathrm{OUT_f}$.

Tab. I. Truth table for a 1-bit full-adder circuit

FA inputs			**FA outputs**	
x	y	c_{in}	z	c_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Tab. II. Surjective/injective mappings found for FA

Member of	**In**	**Out**	**Test**	**Inner Encoding**	**Information stored in the library**
S_f	$\mathrm{xyc_{in}}$	$\mathrm{zc_{out}}$	-	(111,11,0)	$\mathrm{x(0)y(0)c_{in}(0)\vert z(0)c_{out}(0)\vert}$ -
S_f, I_f	x	z	$\mathrm{yc_{in}}$	(100,10,011)	$\mathrm{x(0)\vert z(0)\vert y(0)c_{in}(0)}$
S_f, I_f	y	z	$\mathrm{xc_{in}}$	(010,10,101)	$\mathrm{y(0)\vert z(0)\vert x(0)c_{in}(0)}$
S_f, I_f	$\mathrm{c_{in}}$	z	xy	(001,10,110)	$\mathrm{c_{in}(0)\vert z(0)\vert x(0)y(0)}$
S_f, I_f	$\mathrm{c_{in}}$	$\mathrm{c_{out}}$	xy	(001,01,110)	$\mathrm{c_{in}(0)\vert c_{out}(0)\vert x(0)y(0)}$
S_f, I_f	y	$\mathrm{c_{out}}$	$\mathrm{xc_{in}}$	(010,01,101)	$\mathrm{y(0)\vert c_{out}(0)\vert xc_{in}(0)}$
S_f, I_f	x	$\mathrm{c_{out}}$	$\mathrm{yc_{in}}$	(100,01,011)	$\mathrm{x(0)\vert c_{out}(0)\vert y(0)c_{in}(0)}$

B. Encoding of Transparency Information

For implementation purposes, mapping μ_{INf}:$2^{\mathrm{INf}} \rightarrow \{0,1\}^n$ is defined, assigning a binary string $S_{INf} = (s_{INf}(n\text{-}1)\ldots s_{INf}(0))$ to set $\mathrm{I} \subseteq \mathrm{IN_f}$ in a following way:

$$f_{\mathrm{in}}(i) \in \mathrm{I} \Leftrightarrow s_{INf}(i) \neq 0 \text{ for } i \in \{n\text{-}1,\ldots, 0\}.$$

For FA, let us present only strings $\mu_{INf}(\{\mathrm{x, y, c_{in}}\})=(111)$, $\mu_{INf}(\{\mathrm{x, y}\})=(110)$, $\mu_{INf}(\{\mathrm{x}\})=(100)$, $\mu_{INf}(\{\mathrm{y}\})=(010)$, $\mu_{INf}(\{\mathrm{c_{in}}\})=(001)$ and $\mu_{INf}(\{\})=(000)$ for illustration. Alike, $\mathrm{OUT_f}$ and μ_{OUTf} are defined. Transparency information we are searching for given RTL component described by function f, can be stored as a pair (S_f, I_f), where S_f is set of 3-tuples (*in*, *out*, *test*), each of them coupling μ_{INf}–encoded and μ_{OUTf}–encoded information about inputs ($in \in \{0,1\}^n$) and outputs ($out \in \{0,1\}^k$) among their data a surjection exists under condition of controlling other inputs ($test \in \{0,1\}^n$). Alike, I_f is defined for injection part. For FA, let us present as an example $\{(100,10,011)\} \subseteq S_f$ informing, that a surjection is possible between x-data ("100" on the left) and z-data ("10" in the middle) when controlling y and c_{in} ("011" on the right). For components of wider I/O bit-widths, it seems more efficient to utilize a port names separated by symbol "|" instead of (1…1) notation in case all bits of the port are involved in one of *in*, *out*, *test* members of corresponding 3-tuple. Alike, it is efficient to store single 0 instead of all-zero string 0…0. μ_{INf}/μ_{OUTf}–encoded information is to be stored in the library iff both *in*, *out* members in the 3-tuple are non-zero. For illustration, see Tab. II for all information stored in S_f (contains 7 mappings), I_f (contains 6 mappings) sets.

Above-mentioned, binary-encoded transparency information is suitable for machine processing. To be both practical and easy editable by a human operator, transparency information for each module-type utilized in the circuit structure is stored in a text-file formatted library. For example, the transparency information for FA stored in Tab. II is stored in the library in a following form:

```
MODULE_TYPE FA      // template for module-type FA only
INTERFACE in@x(0) in@y(0) in@cin(0) out@z(0) out@cout(0)
SUR x(0)y(0)cin(0)|z(0)cout(0)|-
INJ
BIJ x(0)|z(0)|y(0)cin(0) y(0)|z(0)|x(0)cin(0) cin(0)|z(0)|x(0)y(0)
cin(0)|cout(0)|x(0)y(0) y(0)|cout(0)|x(0)cin(0) x(0)|cout(0)|y(0)cin(0)
```

In the similar way, transparency information of any digital module can be stored in the library. If transparency information is common to all variants of particular module type (e.g., 1-bit, … n-bit registers) it is possible to make a template for such information. As an example, portion of templatized information for few generally-known module-types follows:

```
MODULE_TYPE REG_<n> // template for module-types REG_1, REG_2, …
INTERFACE in@d(n-1:0) in@clk(0) out@y(n-1:0)
SUR d(i)|y(i)|clk(0)      // compacted version of d(n-1:0)|y(n-1:0)|clk(0)

MODULE_TYPE ADD_<n>cmb    // template for module-types ADD_1cmb, ADD_2cmb, …
INTERFACE in@x[n] in@y[n] out@s(0:n-1)
SUR x(n-1:0)|s(n-1:0)|y(n-1:0)
```

Information for more complex sequential modules can be stored in the library as well – below, cutout of the information for simple scan-register is presented:

```
MODULE_TYPE SREG_<n>
INTERFACE d[n] y[n] clk[1] mode[1] s_in[1] s_out[1]  // a[n] = a(n-1:0)
d(i)|y(i)|clk(0) s_in(0)|y(i)|clk(0)^i
```

Also, special "pseudo-modules" can be constructed by means of the notation. Below, description of pseudo-module JOIN (joining two wires-streams into one wire-strean) and FORK (splitting one wire-stream into two wire-streams) are illustrated:

```
MODULE_TYPE JOIN_<n1_n2>
INTERFACE in@x1[n1] in@x2[n2] out@y(n1+n2:0)
x1(n1-1:0)|y(0:n1-1)|- x2(n2-1:0)|y(n1:n1+n2-1)|-

MODULE_TYPE FORK_<n1_n2>
INTERFACE in@x[n1+n2] in@y1[n1] out@y2(n2-1:0)
x(0:n1-1)|y1(0:n1-1)|- x(n1:n1+n2-1)|y2(0:n2-1)|-
```

All above-illustrated examples can be expressed by means of following *BNF* (*Backus Naur Form*):

```
<lib_element> ::= MODULE_TYPE <mt> INTERFACE <it> <info> <lib_element>
<mt> ::= <identifier> | <identifier>"<"<parameters>">" |
"<"<parameters>">"<identifier>
<parameters> ::= <identifier>|<identifier>_<parameters>
<it> ::= <identifier>@<it_range> |<identifier>@<it_range> <it>
<it_range> ::= [<number>]|(<number>:<number>)
<info> ::= SUR <sur> INJ <inj> BIJ <bij>
<sur> <identifier><map_range> | <identifier><map_range> <sur>
<inj> <identifier><map_range> | <identifier><map_range> <inj>
<bij> <identifier><map_range> | <identifier><map_range> <bij>
<map_range> ::= [<expression>]|(<expression>:<expression>)
```

C. Horizontal Line Test

Our transparency-extraction method is based on efficient implementation of so-called *horizontal line test* utilized in mathematics to determine if a function is injective, surjective or bijective. The principle of the test is as follows. For a graphical visualization of a function f: $\{0,1\}^n \rightarrow \{0,1\}^k$, a horizontal line is constructed for each $y \in \{0,1\}^k$. If f is injective, its graph is never intersected by any horizontal line more than once. If f is surjective, any horizontal line will intersect the graph at least at one point and finally, if f is bijective, any horizontal line will intersect the graph at exactly one point.

Illustration to some of horizontal line tests for FA is presented in Fig 3a-c (*hlxx* are *horizontal lines* needed for the test). Depicted mappings are found unconditionally surjective (Fig 3a) with transparency information written as (111,11,0) or ($x|y|c_{in}$, $z|c_{out}$, 0), bijective when controlling y, c_{in} (Fig 3b), with transparency information written as (001,10,011) or (x, z, $y|c_{in}$) and non-surjective/non-injective (simple) mapping (Fig 3c) as an example of information not to be stored in the library.

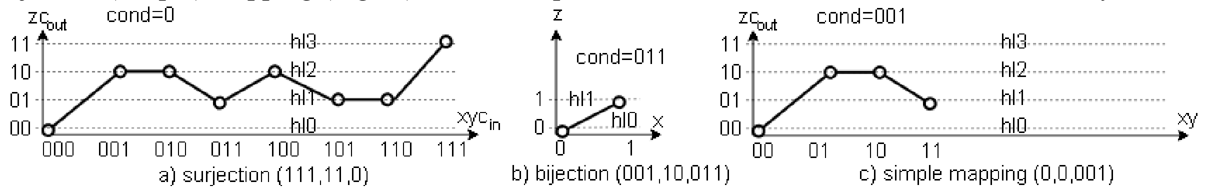

Fig 3 Illustration to horizontal line test

D. Principle, Analysis and Results of Proposed Extraction Method

The space for the paper is limited. So, below the principles of the algorithm for extracting transparency information from true table are presented only.

The idea of S_f –extraction is as follows. For selected constant $c \in |T|$, there are exactly $r = 2^n - 2^{|T|}$ rows with the constant c assigned to particular inputs from T. Because it is tested whether mapping $IN_f \backslash T \rightarrow O$ (where $O \subseteq OUT_f$) is surjective, it is necessary to check, if all $\{0, 1\}^{|O|}$ combinations are assigned to outputs from O. Surely, $I \rightarrow O$ is surjection iff $|I| \geq |O|$ and exactly $2^{|O|-1}$ “1”s and exactly $2^{|O|-1}$ “0”s exist for each output from O. If the condition is not fulfilled, it is necessary to find bits from O making the condition false (i.e., causing some of 2^O combination is missing), remove them from O and test the condition again. This is done while condition is false and O is not empty. It is evident, that up to |O|-1 elements can be removed from O for condition-test purposes.

```
Algorithm 1: Extracting surjections
Input: truth table
Set $S_f$ = {}
Set O = {$f_{out}(k-1)$,…, $f_{out}(0)$}
For each I: $I \in 2^{IN_f} \backslash \{\}$, $|I| \geq |O|$ do
Begin
   Order elements in $T=IN_f \backslash I$ into |T|-tuple $t=(t_{n-|I|-1},…,t_0)$
   Select constant $c \in \{0,1\}^{|T|}$
   Do
      For each true-table row with c assigned to t do S = S ∪ row
      if S'elements represent all $\{0, 1\}^{|O|}$ combinations
         then
            set $S_f = S_f \cup \{(\mu_{IN_f}(I), \mu_{OUTf}(O), \mu_{IN_f}(T))\}$
      else remove problematic elements from O
   While $O \neq \{\}$
End
Output: $S_f$
```

Alike, I_f–extraction algorithm can be constructed. Here, $I \rightarrow O$ is injection iff $|I| \leq |O|$ and each $o \in O$ is mapped-to at most once. If the condition is not fulfilled, it is necessary to find bits from O making the condition false (i.e., causing some of output combinations are mapped-to more than once), remove them from O and test the condition again. This is done while condition is false and O is not empty. It is evident, that maximum of |O|-|I| elements can be removed from O, because removal of more elements from O cause $|I| \leq |O|$ false.

V. Conclusions

In the paper, principle of an automated extraction of diagnostic information from design-related data for testability analysis purposes is described in brief, together with principle of encoding the information. We hope automation of the extraction will contribute to enhancement of digital-circuit design-cycle significantly, especially when designing complex digital circuits. Because of text-file based storing of the information in the library, both further manual correction (creation, addition, deletion, modification) of the information by a designer is still possible. Because for testability analysis purposes, it is not necessary to store so detail data as in case of test-generation purposes, computational complexity of our extraction method is much better than complexity of test-generation related extraction methods, especially in the area of area complexity. For the future research, it is planned to accelerate the search process in a HW, to extend our approach about involving ambiguity information, e.g., f: $\{0,1,X\}^n \rightarrow \{0,1\}^k$ and to deal with other forms of inputs (BDDs, and other representations, which are more compact and thus more practical when comparing to truth-tables) to our extraction method. For example, truth table from Tab. I can be represented by a BDD depicted in Fig 4. We expect new form of inputs will make the extraction process more efficient.

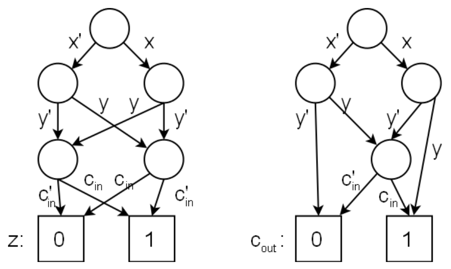

Fig 4 Illustration of BDD for FA

VI. Acknowledgements

The work related to the paper has been financially supported by the *Grant Agency of the Czech Republic* (*GACR*) under post-doctoral contract *GP102/05/P193* "*Optimizing Methods in Digital Systems Diagnosis*" and by the *Research Plan No. MSM, 0021630528 – Security-Oriented Research in Information Technology*.

References

[1] Abadir, M. S., Breuer, M. A.: A Knowledge-Based System for Designing Testable VLSI Chips, IEEE Design and Test of Computers, Vol. 2, No. 4, 1985, pp. 56-68.

[2] Bushnell, M L., Agrawal, V. D.: Esentials of Electronic Testing for Digital, Memory and Mixed VLSI Circuits, Springer,Verlag, 2000, page 129.

[3] Freeman, S.: Test Generation for Data-Path Logic: The FPath Method, IEEE JSSC, Vol. 23, No. 2, 1998, pp. 421-427.

[4] Pečenka, T., Kotásek, Z., Sekanina, L.: FITTest_Bench06: A New Set of Benchmark Circuits Reflecting Testability Properties, In: Proceedings of 9th IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop, Prague, 2006, pp. 285-289.

[5] Makris, Y., Orailoglu, A.: RTL Test Justification and Propagation Analysis for Modular Designs, JETTA, Vol. 13, No. 2, 1998, pp. 105-120.

[6] Makris, Y., Patel, V., Orailoglu. A.: Efficient Transparency Extraction and Utilization in Hierarchical Test. In: Proceedings of the IEEE VLSI Test Symposium, 2001, pp. 246-251.

[7] Murray, B. T., Hayes, J. P.: Test Propagation through Modules and Circuits, In: Proceedings of International Test Conference, 1991, pp. 748-757.

[8] Pečenka, T., Kotásek, Z., Sekanina, L., Strnadel, J.: Automatic Discovery of RTL Benchmark Circuits with Predefined Testability Properties, In: Proceedings of the 2005 NASA/DoD Conference on Evolvable Hardware, Los Alamitos, ICSP, 2005, pp. 51-58.

[9] Pečenka, T., Strnadel, J., Kotásek, Z., Sekanina, L.: Testability Estimation Based on Controllability and Observability Parameters, In: Proceedings of the 9th EUROMICRO Conference on Digital System Design, Cavtat, IEEE CS, 2006, pp. 504-514.

[10] Strnadel, J.: Testability Analysis and Improvements of Register-Transfer Level Digital Circuits, Computing and Informatics, Vol. 25, No. 5, Bratislava, 2006, pp. 441-464.

[11] Vishakantaiah, V., Abraham, J. A., Abadir, M. S. : Automatic Test Knowledge Extraction From VHDL (ATKET), In: Proceedings of DAC, 1992, pp. 273-278.