# TASTE: Testability Analysis Engine and Opened Libraries for Digital Data Path

Josef Strnadel
*Brno University of Technology, Faculty of Information Technology*
*Božetěchova 2, 612 66 Brno, Czech Republic*
*phone: +420 5 4114 1211, fax: +420 5 4114 1270, e-mail: strnadel@fit.vutbr.cz*

## Abstract

*Testability is one of the most important factors that are considered during design cycle along with reliability, speed, power consumption, cost and other factors important for a customer. Estimation of testability parameter strongly depends on how accurate information utilized for the estimation by a testability analysis method is. In the paper, two results of our previous, long-term research in the area of digital circuit testability analysis are summarized: principle of our testability analysis engine and libraries used to store the information outgoing from transparency models. The engine is general and accuracy of its results strongly depends on the information stored in the libraries. If simple transparency model is utilized, information about circuit testability could be far away from real state of the circuit. Otherwise, testability information can approximate so serious parameter such as fault-coverage factor.*

## 1. Introduction

Since reusable components are main building blocks of component-based systems, developing high quality components becomes a critical part of component-based engineering. To generate high quality components from diagnostic point of view, attention must be payed not only to a classical design cycle of each component, but also to a diagnostic related part of the cycle like *synthesis/design for testability* (S/DFT), *test pattern generation* (TPG) etc. For an automation of diagnostic related parts, design-related inputs (library of low-level components, HDL sources, constraints posed on final design) have to be enriched about inputs like *testability analysis* (TA) engine, test patterns and responses and other corresponding data.

In this paper, we will not deal with TPG related parts - they are already well-supported by major *electronic design automation* (EDA) tools. Instead, we will target parts (unjustly) neglected by common EDA tools – a TA engine and its inputs significantly affecting accuracy of TA results and consequently, quality of S/DFT process. Actually, each EDA tool is equipped with its own TA engine that is "hardwired" into the tool. Our idea is to offer a general TA engine that could be utilized by any EDA tool. To be able to do this, we need following: a portable engine (e.g., a code written in ANSI/ISO C++) and EDA tool independent component libraries (for each component, it stores an information making TA process both more efficient and precise and utilizable by any tool).

Many TA approaches have been developed in the past. They can be classified according to several aspects. On the basis of abstraction level, they can be divided, e.g., to gate-level [7][14], *register-transfer level*, RTL, [2][5] [8][9][20][23], *functional level* [4][25], *behavioral level* [22] or *multilevel* [11] approaches. According applicability of results, they can be divided to *general-purpose*, e.g., [2][4][7][22][23] and *special-purpose* (e.g., with results strongly tied to application of a particular DFT technique as the partial scan in [20]). According to a data path analysis utilized, they can be divided to simpler *probability-based* (e.g., [2][5][8][9][14]) dealing with stochastic behavior of a data flow, and exact *structural-analysis* based approaches (e.g., [7][20][23]). In most of the approaches, testability is evaluated by means of controllability and observability parameters. TA approaches only differ in the way, how controllability and observability are defined and measured.

Our TA engine presented in the paper can be classified as multilevel, general-purpose, structural-based TA approach with library-driven behavior and accuracy.

## 2. Our Previous Research

During our previous research activities, we tried to take advantage of so called *transparency principles* utilized for enhancement of hierarchical TPG methods and to utilize them for design of a novel RTL TA engine.

Functionality of the TA engine was experimentally checked in several important areas like

- *S/DFT* area, where a problem of constraint-driven partial/full scan application combined with other techniques was solved by means of evolutionary approaches [23],
- *test-controller synthesis* area, where a problem of test controller synthesis based on testability analysis results was solved [21],
- *synthetic benchmark generation* area [18].

Moreover, in [19], it was experimentally verified that a testability can be estimated very precisely if appropriate conception is utilized for modelling data transfers in a data path – it was shown there is a very close relationship between testability values got by our TA engine (working in linear-close time, which is general assumption posed on any TA algorithm [3]) and fault-coverage values got by commercial TPG tools (generally, working in exponential time) – e.g. for FITTest_Bench06 benchmarks [16], there was only 5% average deviation between testability values gained by our TA engine and fault-coverage values gained by commercial TPG tools.

To be serious, also drawbacks of our TA engine developed for RTL data paths should be mentioned here. First, it worked with fixed library of components (adders, multipliers, multiplexers, registers and substractors). Because the engine was well-optimized for such a type of components, it could generate precise testability results able to estimate commercially gained parameters very closely.

However, it suffered by this advantage – because of a limited number of supported components, it was practically unusable. Second, the library was part of engine's executable. Thus, it was impossible to add new component or to change properties of any component without recoding and recompiling the engine.

## 3. Research Motivation

Both above-mentioned pluses and minuses of our previous RTL engine gave us an enthusiasm for new research, results of which are presented in following part of the paper. Main goal of our new research was to develop general TA engine not limited neither by number and types of components nor to its applicability to RTL data path.

Our idea is as follows: for given general TA engine, its user (not only its vendor) is allowed (in an easy way) to add and/or modify transparency properties of each component and thus affect the quality of TA results related to a particular component and to an analyzed design. In addition to the above-mentioned goal, there is another not-less important goal: portability and usability of both engine and libraries among EDA tools.

## 4. Paper Structure

The structure of the paper is as follows. First, transparency principles are presented. Next, syntax of library language used to describe transparency information for each component is presented together with illustrative examples. After that, a native net-list format (describing circuit structure by means of components from the library) utilized by the engine will be presented together with illustrative examples. Then, a principle of our TA engine is presented in brief, together with results gained by the engine in several areas. At the end of the paper, a brief summary is presented together with possible future research perspectives.

## 5. Concepts Related to Our Approach

### 5.1. Introduction to Transparency

A lot of research efforts have been dedicated to an importance of modeling a data flow in a digital data-path in order to estimate diagnostic parameters of a circuit more precisely.

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 a direction from a *n*-bit port *x* to a *n*-bit port *y* in a circuit data path iff an one-to-one (i.e., bijective) mapping exists between *x*-data and *y*-data; in such a case, so-called *i-path* exists (in the direction) from *x* to *y*. As the first illustrative example, see Fig. 1a) – an i-path from port *a* to port *y* of a multiplexer MX exists iff MX's selection input *sel* is set to 0.

Alike, in Fig. 1b) – an i-path from *d* to *y* of register R exists iff a rising edge appears on R's *clk*.

In Fig. 1c), an i-path from *a* to *y* of adder ADD exists iff ADD's *b* is set to all 0's.

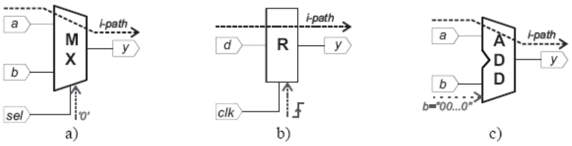

**Fig. 1. Illustration to i-paths**

Other works tried to enhance properties of the above-mentioned model. E.g., in a conception referred to as *S/F-Path* conception [5], it was shown the I/T-Path conception is easy to understand and implement, but it is too strict in definition of a data path suitable for diagnostic data-flow – I/T-Path's strict "bijective mapping requirement" leads to an unneeded restriction of a data path portion suitable for transferring diagnostic data (especially when port bit-widths in a component interface differ or some bits in interface are uncontrollable and/or unobservable).

The strict requirement can be soften by *analyzing a data path separately* for transferring test vectors/patterns (responses) between *x* and *y*. The idea of such a separate analysis is as follows: test vectors/patterns (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 greater data path portion can be considered suitable for diagnostic data flow than in a case of I/T-Path conception.

Several variations of the above-mentioned approaches have been used in the area of generating so-called *hierarchical tests*: ambiguity sets [7], transparency modes [11], or transparency channels [5, 6] etc. All of the 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.

## 5.2. Language Utilized in the Library

Actual syntax of a language utilized by our TA engine to describe transparency properties of components can be expressed by means of *Backus Naur Form* (BNF):

```
<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>|<identif.><map_range><sur>
<inj>:=<identifier><map_range>|<identif..><map_range> <inj>
<bij>:=<identifier><map_range>|<ident.r><map_range> <bij>
<map_range> ::= [<expression>]|(<expression>:<expression>)
```

By means of the above-described language, it is possible to assign all bijective, surjective and injective mappings to each component. E.g., for a generally known 1-bit full adder (FA) with an interface depicted in Fig. 2, information stored in the library could be written in a following way:

```
MODULE_TYPE FA1
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)
```

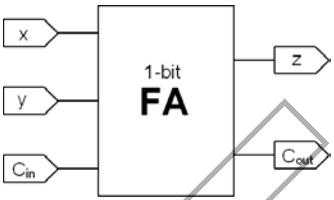

**Fig. 2. One-Bit Full-Adder Interface**

Let us explain a meaning of the above-mentioned information now. In the first line, component type (FA1) is identified after the "MODULE_TYPE" keyword.

In next line, an interface of FA1 is defined after "INTERFACE" keyword, i.e., an input bit 0 allocated for a port x (can be also written as in@x[1], which means an input port x of bit-width equal to 1) etc.

After "SUR", "INJ" and "BIJ" keywords, information about surjective, injective and bijective mappings among particular interface bits is stored. Each part of the information is in the "s|t|c" form consisting of sub-parts s, t, c separated by "|" character, where "s" is a mapping source (domain), "t" is a mapping target (co-domain, image) and "c" is a condition required to enable the mapping.

Above-mentioned example describes transparency information for one particular component only (1-bit FA). This is the simplest way of utilizing the library. However, by means of the language, it is possible to describe a more general information covering entire class of circuits – e.g. *n*-bit registers or adders with interfaces depicted in Fig. 1a, Fig. 1b. Below, only part of the information is given:

```
MODULE_TYPE R_<n>  // one template for R_4, R_32, …
INTERFACE in@d(n-1:0) in@clk(0) out@y(n-1:0)
SUR d(n-1:0)|y(n-1:0)|clk(0)

MODULE_TYPE ADD_<n>a // ADD_4a, ADD_16a,  …
INTERFACE in@a[n] in@b[n] out@y(0:n-1)
SUR a(n-1:0)|y(n-1:0)|b(n-1:0) b(n-1:0)|y(n-1:0)|a(n-1:0)
```

## 5.3. Net-list Format

After transparency-related information is stored in the library for each component-type or class of component-types, it can be assigned to particular components design consists of.

The structure of particular design can be described in native net-list format, of which syntax can be expressed by means of following BNF (simplified because of space limit for the paper):

```
<design> ::= DESIGN <identifier> <it> <com> <link>
<it> ::= <identifier>(<port_type>)|<identif.>(<port_type>) <it>
<port_type> ::= in|out|inout|clk|sel
<com> ::= COM <identifier> (<com_type>)|
          <identifier>(<com_type>) <com>
<link> ::= LINK <identifier> -> {<identifier>}|
          <identifier> -> {<identifier>} <link>
```

As an example, let us present the net-list describing the circuit depicted in Fig. 3 (let us suppose component-types ADD_<n>a, MUL_<n1, n2> and REG_<n> are stored in the library):

```
// C++ comments supported
//
// design name and interface:
DESIGN nl in(in,4) clk(clk,1) out(out,4)
// components within the design (interfaces and properties are
given // by component-type):
COM ADD1(ADD_4a) // component of type ADD_<n>a
COM ADD2(ADD_4a) // component of type ADD_<n>a
COM MUL1(MUL_4_8comb) //c. of type MUL_<n1, n2>comb
COM R1(REG_4)     // component of type REG_<n>
COM R2(REG_4)     // component of type REG_<n>
COM R3(REG_4)     // component of type REG_<n>
// physical connections among bits in component interfaces:
LINK nl.in(3:0) -> ADD1.a(0:3) //explicit wiring
LINK nl.clk -> R1.clk  // implicit full-width connections
LINK nl.clk -> R2.clk
LINK nl.clk -> R3.clk
LINK ADD1.y -> R1.d
LINK R1.y -> ADD2.a
LINK ADD2.y -> R2.d
LINK R2.y -> MUL1.a
LINK MUL1.y(7:4) -> R3.d       // source bit-range selection
LINK R3.y -> ADD1.b ADD2.b MUL1.b nl.out(3:1) nl.out(0)
// multiple full or partial bit-width targets and destinations
allowed
```

## 5.4. Principle of our TA Engine

Having the information (stored in the net-list) about how interfaces of various modules are interconnected and information (stored in the library) about mapping of data between each pair of adjacent ports in a data-path, it is possible to construct two special digraphs for the data-path:

- *test pattern data-flow digraph* $G_S = (V_S, E_S)$ and
- *test response data-flow digraph* $G_I = (V_I, E_I)$.

A set of vertices of $G_S$ ($G_I$) consists of ports. An oriented edge exists between two vertices iff a surjection (injection) exists between the start-vertex and end-vertex data (i.e., iff it is possible to transfer test vectors (responses) in the direction from the start-vertex to end-vertex). Also, each edge in $G_S$ ($G_I$) is evaluated by a "transfer-condition" function $\mu_E$: $E \rightarrow 2^V$, where $E = E_S \cup E_I$, $V = V_S \cup V_I$. Using $\mu_E$, set of ports necessary to control an edge $e \in E$ is assigned to the edge (see Fig. 5 as an example).

Let us suppose now that modification of the NL circuit (Fig. 3) according to Fig. 4 (the multiplexer MUX1 is added to a data path between the MUL1.y(7:4) and R3.d in order to enhance testability of NL by breaking the most-nested loop). Let us denote modified circuit as NL*.

In Fig. 5, portion of $G_S$ (Fig. 5a) for adjusting test data from the PI tst_in to the input b of the ADD1 (ADD1.b) is presented together with a portion of $G_I$ (Fig. 5b) for observing test data from the output y of the MUL1 (MUL1.y) at the PO out. In the figure, following graphical notation is used. In full-line circles, ports of in-circuit components are depicted, in dash-line circles, PIs/POs are depicted and in a double-line circle, port the digraph portion belongs to is depicted. Circles connected by a full-line represent test path for the double-lined port and circles connected by dash-line represent paths to be controlled in order to ensure the data flow through full-line path. Each edge is evaluated by means of $\mu_E$.

Proposed TA algorithm is constructed as a graph-searching algorithm over $G_S$ and $G_I$ [23]. During the search process, accessibility of ports from PIs is analyzed and evaluated in $G_S$ (controllability analysis step) first and after that, accessibility of ports at POs is analyzed and evaluated in $G_I$ (observability analysis step).

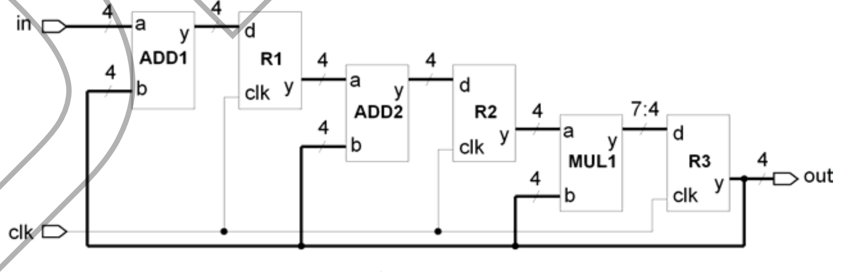

**Fig. 3. NL circuit**

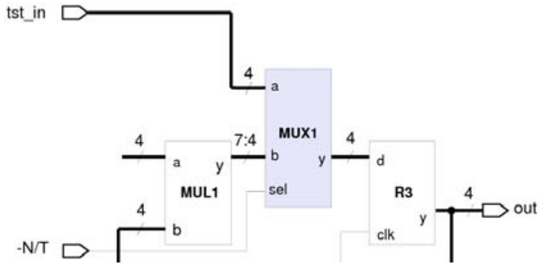

**Fig. 4. Modification of NL circuit**

For $G_S$ analysis, mapping $\mu_S$: V → $(V \cup \{\square\}) \times \mathcal{R}$ is defined assigning an ordered tuple (p*, c*) to an input port *p*, where

- *p** is a set of ports, from an accessibility mark has been propagated for an input port *p* and
- *c** is controllability value of the mark after its propagation.

In an informal way, principle of propagating marks through $G_S$ can be described in a following way:

[**Initialization**] Assign marks to primary input vertices of $G_S$; leave other nodes markless.
[**Edge selection**] Select such edges from $G_S$, whose start-vertex together with edge condition inputs are marked and whose end-vertex is either markless nor it has worse mark than the mark to be transported; continue with step 3. If there is no such an edge, finish the algorithm.
[**Marking**] Assign a mark to end-vertices of selected edges and go to step 2.

Alike, mapping $\mu_I$ (as well as propagating mechanism) exist for $G_I$ analysis. As an example for above-depicted circuit, accessibility marks are serially propagated through edges in $G_S$ in a following way ("*a* → *b* (*c*)" notation means propagation of accessibility mark with controllability value *c* through an edge (*a*, *b*) from $E_S$):

NL*.in → ADD1.a (1.0),
NL*.tst_in → MUX1.a (1.0),
NL*.sel → MUX1.sel (1.0),
NL*.clk → R3.clk (1.0),
NL*.clk → R2.clk (1.0),
NL*.clk → R1.clk (1.0),
MUX1.a → MUX1.y (0.989),
MUX1.y → R3.d (0.989),
R3.d → R3.y (0.733),
R3.y → NL*.out (0.733),
R3.y → MUL1.b (0.733),
R3.y → ADD2.b (0.733),
**R3.y → ADD1.b (0.733)**,
ADD1.b → ADD1.y (0.670),
ADD1.y → R1.d (0.670),
R1.d → R1.y (0.497),
R1.y → ADD2.a (0.497),
ADD2.b → ADD2.y (0.733),
ADD2.y → R2.d (0.332),
R2.d → R2.y (0.219),
R2.y → MUL1.a (0.219),
MUL1.a → MUL1.y (0.146),
MUL1.y → MUX1.b (0.146).

Because of limited space in the paper, illustrative examples are presented instead of formal description of the TA algorithm. Worst-case time complexity of the algorithm is $O(|V(G_S)| \times |E(G_S)| + |V(G_I)| \times |E(G_I)|)$, i.e. quadratic.

However, the complexity can be achieved only to very small class of synthetic circuits. For practice circuits, it is usually reduced to better linear time complexity (see Fig. 6 depicting experimentally measured average execution time as a function of number of components in analyzed designs from [18] synthetic benchmark suite).

## 6. Experimental results

In this section, possible applications of results achieved by proposed TA engine are presented.

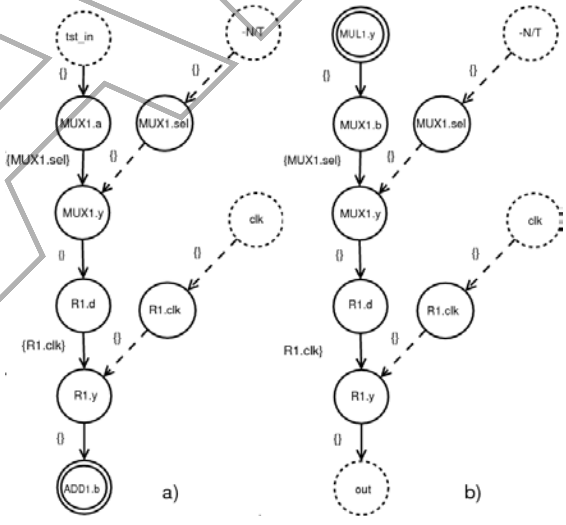

**Fig. 5. Illustration to $G_S$**

### 6.1. Fault-Coverage Estimation

It can be said TA is utilized in the areas where information about testability is required in pre-defined limited time and accuracy. If a precise information is needed, a TPG tool can be applied to a design in order to produce, e.g., a fault-coverage parameter of the design. However, TPG algorithms are of exponential time complexity in general. Problem arise if an information is to be evaluated, e.g., in each iteration of design space exploration algorithm in order to evaluate quality of a particular solution from testability point of view. In such a situation, a TPG algorithm would be too expensive. To solve the problem, fault-coverage parameter can be estimated (ideally, in linear-close time), e.g. by means of a TA method. To verify applicability of our TA results for fault-coverage estimation purposes, we have experimentally measured an average deviation between results gained by our TA engine and fault-coverage results gained by the commercial TPG tool (FlexTest from MentorGraphics company).

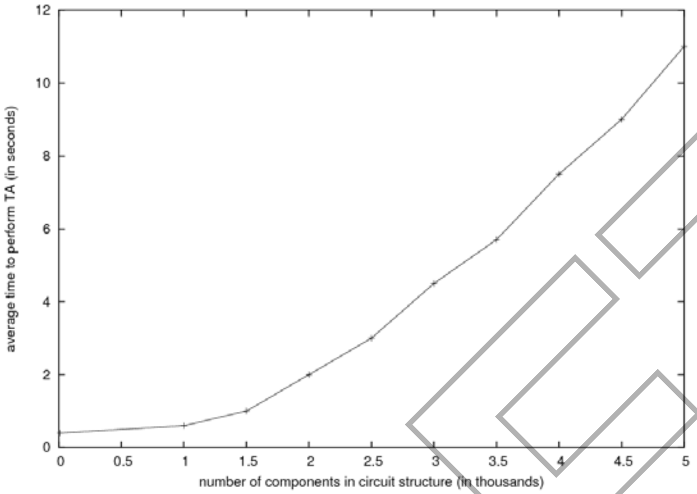

**Fig. 6. Experimentally measured time-complexity of TA**

After a deep analysis of FITTest_Bench06 benchmarks [16], there was only 5% average deviation between testability results (see Fig. 7). Of course, we do not expect this will hold generally for all classes of circuits, so further experiments are needed in this area.

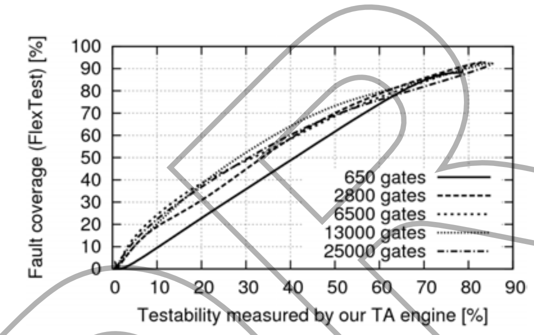

**Fig. 7. TA results utilized for fault-coverage estimation**

## 6.2 Design Space Exploration Results

In the following, an applicability of TA results in S/DFT areas is presented in brief.

In S/DFT, it is necessary to explore the search-space of possible solutions in order to be able to discover the solution with properties as close as possible to desired (optimal) properties. Because it is necessary to find out an optimal or optimum-close solution, one of the most important problems that should be solved is the problem of evaluating quality of the solutions.

Usually, the evaluation is done by means so called *fitness function* (*fit*). The higher *fit* is for particular design given as its parameter, the closer the design is to desired optimum. Because testability is a very important factor in S/DFT, *fit* is evaluated by means of TA results. Of course, there can be extra requirements posed on final design like maximal allowed area/pin overhead caused by S/DFT application, maximal allowed power consumption etc. Also all those factors play role during *fit* evaluation. Fitness function utilized in our approach can be expressed by following formula:

$$fit = \frac{tst_{act}}{(1 + |ao_{act} - ao_{constr}|) \times (1 + |po_{act} - po_{constr}|)}$$

,

where $ao_{constr}$ ($po_{constr}$) are maximal area (pin) overheads to be payed for testability enhancement of the original circuit structure and $ao_{act}$, $po_{act}$, $tst_{act}$ denote area overhead, pin overhead (both in % divided by 100) and testability (real number from <0.0; 1.0> interval) values of particular solution from the state space. E.g., DFT process driven by means of fitness values given by means of the above-mentioned formula, can be expressed by the following steps:

1. [**DFT implementation**] new solution with built-in DFT techniques is generated
a) [**Selection of DFT modification**] particular configuration based on user-selected DFT techniques is generated randomly (special mutation of previously gained solutions is utilized for the purpose)
b) [**Implementation of DFT modification**] configuration from point 1a) is built into the origina circuit structure
2. [**Evaluation**] fitact value iassigned to the solution
3. [**Detection**] if fitact > fitbest, the best solution is found
4. [**DFT removal**] DFT techniques are removed from the circuit structure

In Fig. 8, result of the following DFT experiment is presented. The objective of the experiment was to check whether solutions with lower $ao_{act}$, $po_{act}$ values and higher $tst_{act}$ value are assigned higher fit value than solutions with higher values of overhead parameters and lower value of testability parameter.

It can be seen that solutions evaluated by higher *fit* values are those with good cost/quality trade-off between costs of S/DFT application and testability enhancement achieved by the application. As an example of such solutions, see values for iterations 683, 716 or 721. As an ”opposite” example, see values for iterations 687, 688 or 703.

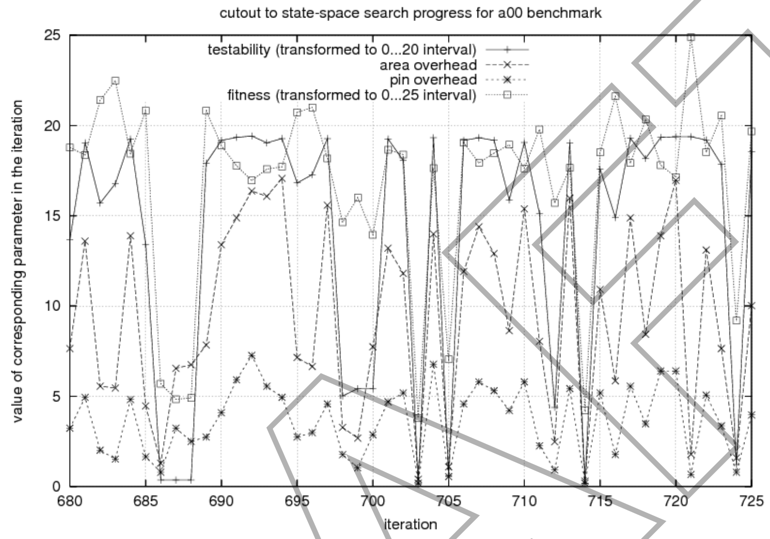

**Fig. 8. Cutout of DFT search space exploration for a00 benchmark**

- A great diversity of quality (measured by means of *fit* values) of solutions generated during the search process can be achieved, which is important for S/DFT ability to find solutions with high cost/quality trade-off. Other experiments showed 1) time S/DFT needs to find the solution for particular benchmark is proportional to the complexity of a circuit, 2) for all tested benchmarks ,”average-quality” solutions “quickly” while it takes much more iterations to find optimum-close solutions.

## 7. Conclusion

In the paper, principles of our TA engine, together with syntax and illustrative examples of related component-libraries were presented in brief. Main goal of the paper was to present TA engine usable by any EDA tool and applicable to common S/DFT areas. Comparing to its predecessor, the engine is portable (implemented in ANSI/ISO C++) and not limited neither by number and types of components nor to its applicability to RTL data path. There are two inputs required by the engine: library of components and net-list. Both of the inputs are stored in separate plain-text files and thus can be easily modified by both human and machine.

Library of components (containing information about interfaces and transparency properties of particular components) can be provided by a component vendor or can be created or modified by a user. It is important to know that accuracy of TA results is significantly affected by accuracy of information stored in the library.

Second input (the net-list) is utilized to describe inter-connections between interfaces of particular component instances involved in the design. The net-list can be created manually by a user or it can be easily generated, e.g., from VHDL, Verilog or EDIF file utilized by an EDA tool. Actually, TA results produced by our TA engine can be stored in a plain-text file, TEX file and HTML file to be easily published by common publishing systems. Extension to further file-formats is possible in a simple way. Our further research will be dedicated especially to TA and S/DFT of hierarchical and system-on-a-chip (SOC) digital and mixed-signal designs, which belong to the most popular approaches at present. Also, further experiments are planned.

The work related to the paper has been financially supported by the Research Plan No. MSM, 0021630528 – *Security-Oriented Research in Information Technology*.

## 9. References

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

[2] P., Bukovjan: Allocation for Testability in High-Level Synthesis. PhD thesis, Institute National Polytechnique de Grenoble, 2000, 124 p.

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

[4] C. H., Chen, P. R., Menon: An Approach to Functional Level Testability Analysis. In: Proceedings of the International Test Conference, 1999, pp. 373—380.

[5] J., Fernandes, M. B. Dos Santos, A. Oliveira, J. P. Teixeira and R. Velazco: Sensitivity to SEUs Evaluation using Probabilistic Testability Analysis at RTL, In: Proceedings of 8th LATW, Cusco, 2007, 6 p.

[6] S., Freeman: Test Generation for Data-Path Logic: The FPath Method, IEEE JSSC, 23(2), 1998, pp. 421-427.

[7] L. H. Goldstein and E. L. Thigpen. SCOAP: Sandia controllability/observability analysis program. In DAC '80: Proceedings of the 17th conference on Design automation, pp. 190–196. ACM Press, 1980.

[8] J., Grason: TMEAS - a Testability Measurement Program. In Proc. of IEEE/ACM Design Automation, 1979. pp. 156–161.

[9] X., Gu: RT Level Testability Improvement by Testability Analysis and Transformations. Ph.D. Thesis, Linköping University, 1996. 160 p.

[10] V. I. Hahanov, M. A. Kaminska and O. Lavrova: Testability Analysis of the VHDL Structure for Fault Coverage Improving, Electronics and Electrical Eng., Vol. 74, No. 2, 2007, pp. 29-32

[11] J., Hlavička, Z., Kotásek, R., Růžička and J., Strnadel: Interactive Tool for Behavioral Level Testability Analysis, In: Proceedings of the IEEE ETW 2001, Stockholm, SE, 2001, pp. 117-119.

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

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

[14] C. M., Maunder, R. G., Bennetts and G. D., Robinson: CAMELOT: A Computer-Aided Measure for Logic Testability. In Proceedings of Intenational Conference on Computer Communication, 1980. pp. 1162–1165.

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

[16] 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.

[17] T., Pečenka: Tools and Methods for Automated Generating of Benchmark Circuits. PhD thesis, Brno University of Technology, 2007. 124 p.

[18] T., Pečenka, Z., Kotásek, L., Sekanina and J., Strnadel: 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.

[19] T., Pečenka, J., Strnadel, Z., Kotásek and L., Sekanina.: 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.

[20] R., Růžička: Formal approach to the Testability Analysis of RT Level Digital Circuits. PhD thesis, Brno University of Technology, 2002, 102 p.

[21] R., Růžička, J. Strnadel.: Test Controller Synthesis Constrained by Circuit Testability Analysis, In: Proceedings of 10th Euromicro Conference on Digital System Design, Architectures, Methods and Tools, Los Alamitos, US, ICSP, 2007, pp. 626-633

[22] S. Seshadri and M. Hsiao. Behavioral-level dft via formal operator testability measures. Journal of Electronic Testing, 18(6):596–611, 2002.

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

[24] V. M., Vedula and J. A., Abraham: FACTOR: A Hierarchical Methodology for Functional Test Generation and Testability Analysis. In Proceedings of Design, Automation and Test in Europe Conference and Exhibition, IEEE, 2002. pp. 730–735.

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