STRUCTURE-PRESERVING MODELLING OF REAL CIRCUITS BASED ON SIGNAL FLOW GRAPH

Mohamed Denguir and Sebastian M. Sattler

Friedrich-Alexander-University Erlangen Nuremberg (FAU)
Chair of Reliable Circuits and Systems (LZS), Erlangen, Germany

ABSTRACT

For the functional safety of safety-critical circuits and systems, the function of the real structure has to be modelled in a structured manner. It must therefore be ensured that the formally derived and modelled function is consistently conformed to the function generated by the real structure. Conversely, it is fatal for safety if the modelled function has a behaviour, which differs from the realized function. For the first time, this paper introduces a structure-based verification method using signal flow graphs at lattice level. A digital circuit will make this more comprehensible and the results will be presented in signal flow graphs. Based on the results of our verification method, we developed a structure-preserving method for fault diagnosis and localization. The presented verification method leads to the creation of a quaternary vector list as a database. This database encodes the phase lists of the weighted edges of a signal flow graph. It is possible to program search functions, going through the database and determining certain criteria, e.g. test coverage, fault coverage and test accuracy. However, the lists generated by a complex reality can assume a great extent in their dimensions. In order to achieve a lower memory requirement and a shorter computing time, the database should be compacted without loss of information. In this article, a self-developed method for data compaction is additionally presented and exemplified.

KEYWORDS

Formal Verification of hardware, Fault Modelling, Fault Diagnosis, Structure-Preserving Modelling, Testing of Digital electronic circuits, Specification, Test Compaction

1. INTRODUCTION

Simulation-based validation methods are the most frequently used tools for validating complex circuits and systems. However, since design and technology are increasingly limited, simulation-based validation is not capable of covering all possible scenarios, which is an unacceptable basis, especially for safety-critical systems [1]. It is therefore known that widespread simulation techniques are not sufficient to ensure the correctness of complex systems [2]. Consequently, the cost of testing complex systems, with which all scenarios need to be covered, has increased for why test is becoming a dominant factor in overall production cost [3]. The alternative is to use theoretically well-founded formal verification tools [4], whereby the exact allocation of virtual functionality and real structure takes place. It is therefore useful to use a robust and effective verification method that tests the functionality of complex and directed real systems. In this paper, there we present a formal method named "Structure-Preserving Modelling based on Signal Flow Graph" for verification and testing of circuits and systems. We first introduce the structure-preserving methodology we propose by explaining the steps and the underlying rules. We illustrate the method on a real, simple circuit with self-injected faults. A structure-preserving method for fault diagnosis and localization is presented and used on our circuit. Finally, we present a method and its realization for loss-free data compression, in order to achieve a compacted test process.

2. STRUCTURE-PRESERVING MODELLING

We are interested in formal, theoretical modelling methods that cover digital and analog properties of a given structure (system or circuit) and support model-based verification for system

behaviour. Furthermore, the theory should be equipped with the ability to subdivide an arbitrary structure into substructures and to compose the resulting sub-models into an overall model. The modelling of real systems by the use of functions (using the description of the behaviour) that preserve the structure, leads to a presentation in a signal flow graph (SFG). A SFG is the presentation of an abstract algebra, the formulas that can be used to verify structure-preserved images. The SFG is the directed graphical presentation of a multiple set (multi-set), which consists of so-called edges and nodes, which represent morphisms (e.g. functions) and objects (e.g. sets). The phase lists of a SFG consisting of nodes and edges are independent from one another and without restriction of the generality concurrent to each other (simultaneously in parallel) [5]. This data model in positive logic (PL) with additional properties like associativity and identity forms the prerequisites for encoding a control circuit [6]. This data model splits into Operation (OP) and Control (CTRL) and guarantees the functions specification (spec), test (test), functionable (k) and non-functionable (/k) for each partial structure.

2.1. STEPS TO CREATE THE DATA MODEL

The creation of the data model is divided into three steps: First (1), the real pins that occur in the real structure must each be designated with a positive or negative literal, which corresponds to the embedding of the real structure into a coding universe. Then (2), the real structure is abstracted in an event-based manner and simulated in a model by a directional signal flow using two paths (dual-rail). Subsequently (3), the state transitions of the sub-models are described in propositional logic expressions (AA), their SFG are encoded with (1, 0, -) in three-valued logic (so-called ternary vector list, TVL [7]) and formatted as quaternary vector list (QVL) in (1, 0, -, X).

2.2. RULES FOR GENERATING THE MODEL

The first step towards the creation of the data model, the labelling of the pins, can be channelled with the "directed" laws of the respective physics or electrical engineering by means of two rules. First and most important rule: naming pins according to pin partitioning [5], these are states (S), primary inputs (PI) and primary outputs (PO). The second rule is to fine-tune the pins according to the transmitted digital signal value (1 or 0), negative literal for signal value equals 0 (low) and positive literal for signal value equals 1 (high). The negative literal is marked with the prefixed symbol "/". Regarding the second step for creating the data model, the representation of the "substitutable" complementarity by means of the operation switch (¬) is carried out in dual rail [8]. The rules for carrying out the third step for creating the data model, the coding of the functions spec, test, k and /k in TVL are: spec and test require defined limits and fault models to be known, k and /k require realities that are e.g. components and lines. As each state is splitted into two states, called present state S^a and next state S^n, $S = (S^a, S^n)$ is generally determined in that way, that OP spec and test are directed from S^a to S^n, while CTRL k and /k are directed from S^n to S^a. Thus, as soon as spec is fulfilled, a pin or a state retains its value in terms of a state stabilization [9], $S^n = S^a$. By contrast, if test is fulfilled (fail), the state changes into its complementary (substitute), i.e. into the same state with negative literal, $S^n = /S^a$. Since k and /k depend on their respective reality, it is inconvenient to develop a general formula for their fulfillment. They are modelled explicitly or implicitly.

2.3. COMMON EXAMPLE

The particular data model is generated from the following design pattern. Let X and Y be two real pins, that are present in a structure, then its associated SFG is shown in Figure 1 (left). Real pins X and Y has been here (e.g.) declared as positive literals X and Y. The SFG shown in Figure 1 (left) can be reduced to the SFG in Figure 1 (right), $X = (X^a, X^n)$ and $Y = (Y^a, Y^n)$. Table 1 shows the state transitions as phase lists. The state transitions (events) from defined limits, reality and known errors are encoded in TVL. The star symbol "*" represents "undefined" regarding the entire article. Is a list place reserved with the symbol "∗", it means that this place does not exist.

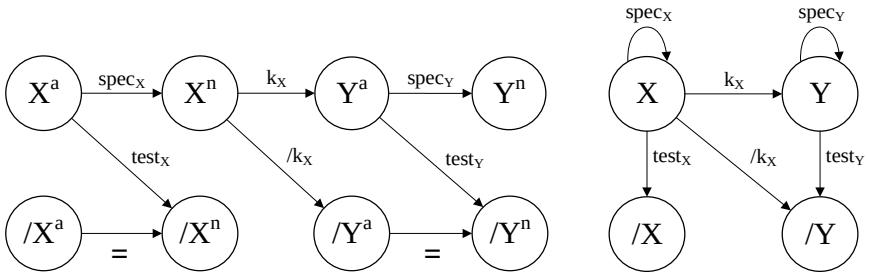


Figure 1. Common data model - Reduced data model (in PL)

Table 1. State transitions and functions (in TVL)

S^a		Event			S^n		Function	
X	**Y**	**defined limits**	**real world**	**known faults**	**X**	**Y**	**OP**	**CTRL**
1	*	(1,0,-)	*	*	1	*	$spec_X$	*
*	1	*	(1,0,-)	*	1	*	*	k_X
1	*	*	*	(1,0,-)	0	*	$test_X$	*
*	0	*	(1,0,-)	*	1	*	*	$/k_X$
*	1	(1,0,-)	*	*	*	1	$spec_Y$	*
*	1	*	*	(1,0,-)	*	0	$test_Y$	*

3. EXAMPLE DIGITAL CIRCUIT

In this chapter, a real digital circuit (DUT - Device-Under-Test) should be prepared for the verification of known, self-defined errors with the help of "Structure-Preserving Modelling based on SFG". We consider the digital circuit shown in Figure 2, which is already available on a real board. The circuit is equipped with jumpers, which serve as well-known faults to be modelled (injected). The goal we pursue is to carry out the verification on the basis of deliberately installed errors by plugging in and removing jumpers, respectively. For this purpose, an automatic test device based on a µ-Controller is developed and programmed. In this article, we will restrict ourselves to the theoretical part, the report on structure-preserving modelling of a circuit.

3.1. SCHEMATIC REPRESENTATION OF THE DUT

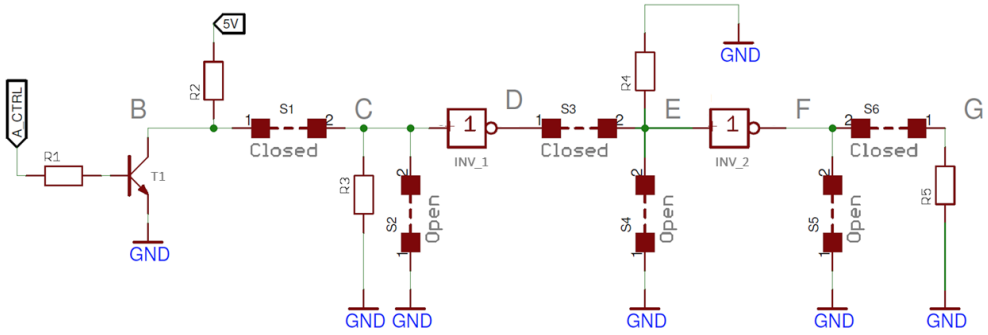


Figure 2. Schematics of DUT (by courtesy of Liebherr GmbH)

The digital circuit in Figure 2 shows CMOS inverters (INV_1 and INV_2) connected in series and controlled by a BJT (npn). The circuit includes six switches S1, S2, S3, S4, S5 and S6, serving as jumpers and which can be plugged in (closed) or out (open) manually. In order to avoid any undesirable electrical interruptions or short circuits in normal operation, switches S1, S3 and S6 are closed, while switches S2, S4 and S5 are open. If a digital 1 is set at the input (A_CTRL), transistor T1 becomes conductive and pin B is pulled to GND (digital 0). If switches S1 and S2 are in normal operation, pin C takes this digital 0. This is then inverted by the inverter

(INV_1) and outputted as digital 1 to pin D. Pin E accepts the digital 1 from pin D if the switches S3 and S4 are switched to normal operation. This is then inverted by the inverter (INV_2) and outputted as digital 0 to pin F. Pin G at the output accepts the digital 0 from pin F if the switches S5 and S6 are in normal operation. Otherwise, if digital 0 is applied to the input, the transistor T1 is non-conductive and pin B takes the digital 1 out of the ohmic resistors R2 and R3 $\gg$ R2 due to the voltage divider. The value is switched to digital 0 by the inverter (INV_1) and transferred to pin E. Resulting of inverter (INV_2) the value is switched to digital 1 and propagated to pin G.

3.2. STRUCTURE-PRESERVING MODELLING OF THE DIGITAL CIRCUIT

3.2.1. Designate the pins

It is desired to get a digital 1 at the output, but for that digital 0 must be applied to the input (A_CTRL). Therefore, after taking the rules in section 2.2 into account, the input pin A_CTRL at the BJT-base must be declared as negative literal /A and described as PI. Consequently, after consideration of the reality (section 3.1), the real pins B, C, F and also G must be declared as positive literals B, C, F and G and the pins D and E as negative literals /D and /E. It results in six pins or states (B, C, /D, /E, F, G) and one PI (/A). Pin G is both a state and PO. The switches are declared as positive literal when closed in normal operation, otherwise with negative literal. The list is S1, /S2, S3, /S4, /S5 and S6.

3.2.2. SFG in Dual-Rail

After successfully implementing the first step of the verification method, the designating of the pins, we can now apply the second step, abstracting the reality in event-based manner, and present a SFG in dual-rail using the operation switch (¬). By the operation switch, the six original states (B, C, /D, /E, F, G) become six substitutable complementary states (¬B, ¬C, ¬/D, ¬/E, ¬F, ¬G). On each of the twelve states the function test is applied, which means that twelve substitutable, complementary states can be achieved. In summary, the states B, C, /D, /E, F and G result in /B /C, D, E, /F and /G and from ¬B, ¬C, ¬/D, ¬/E, ¬F and ¬G results /¬B, /¬C, /¬/D, /¬/E, /¬F and /¬G. Figure 3, shows the SFG. The state (5V, /GND) in the SFG represents the voltage supply. The transition from state (5V, /GND) to state B and dual to that to state ¬B follows by functionable k_{5V} und $k_{/GND}$.

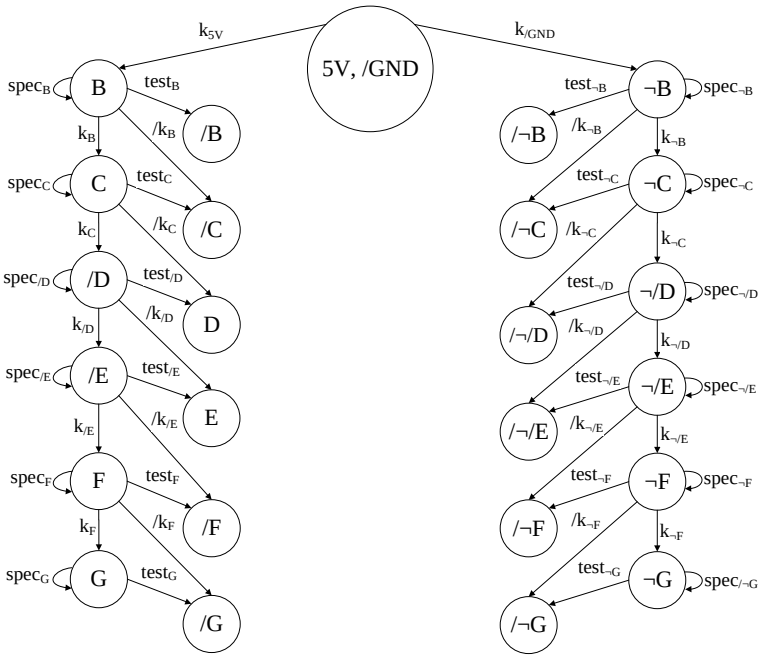


Figure 3. Data model (SFG) of the DUT

3.2.3 Coding of the functions (OP, CTRL)

3.2.3.1 Embedding in the coding universe

Before we proceed to create the data model with the assigning (encoding) of the state transitions spec, test, k, and /k in TVL in the third step, we depict the circuit in a complete value table (with all input combinations). The following nine value tables (Table 2) result. The creation of these complete value tables, or their listing as TVL, is mandatory and serves to control the results in the third step, which is carried out in coming section.

Table 2. Circuits as value table (in TVL)

B	/A	S1	/S2	B
-	0	-	1	1
-	0	0	-	1
-	1	-	-	0
-	0	1	0	0

B	S1	/S2	C
1	1	1	1
1	-	0	0
1	0	-	1
0	-	-	0

C	S1	/S2	C
1	1	1	1
-	-	0	0
-	0	-	0
0	1	1	0

/D	S3	/S4	/D
0	-	-	0
1	1	0	0
1	0	-	1
1	1	1	1

/D	S3	/S4	/E
0	-	-	0
1	0	-	0
1	-	0	0
1	1	1	1

/E	S3	/S4	/E
-	0	-	0
-	-	0	0
0	1	1	0
1	1	1	1

F	/S5	S6	F
1	1	-	1
1	0	-	0
0	-	-	0

F	/S5	S6	G
1	1	1	1
1	-	0	0
1	0	-	0
0	-	-	0

G	/S5	S6	G
1	1	1	1
1	-	0	0
1	0	-	0
0	-	-	0

3.2.3.2 Coding of the function in TVL

The following coding-tables in Table 3 to Table 8 serve together with their SFG as a more detailed description of weighted edges of the SFG from Figure 3. They exhibit the specific encoding of the OP-functions (spec, test) and CTRL- functions (k, /k). This is done, after taking the rules from section 2.2 into account. Analogous to the structure of Table 1, $\delta(S^a, S^n)$ depends on the states (B, C, /D, /E, F, G), the PI (/A) and the fault models (S1, /S2, S3, /S4, /S5, S6). In Table 3, the state B retains its digital 1 if the function $spec_B$ is fulfilled. This is done without influence of the errors, i.e. (S1, /S2) = [∗ ∗], the value of the PI /A is set to digital 0. If the function $test_B$ is fulfilled, state B changes to digital 0. This is due to (/A, S1, /S2) = [0 1 0] and (/A, S1, /S2) = [1 - -]. In the case of the first combination (S1, /S2) = [1 0], a known fault is detected as present, switch S2 is closed (electrical short circuit). In the other case, (S1, /S2) = [- -], it could occur (S1, /S2) = [1 1], causing the known fault to be correctly recognized as not present. In this way, the data model contains all the answers to known errors detected as existing faults and known errors detected as non-existent faults. Since power supply is always assumed to exist, k_{5V} and $k_{/GND}$ are considered as fulfilled and do not require coding (related SFG to Table 3). Otherwise, k_B, $/k_B$ and $k_{\neg B}$ are explicitly modelled in Table 4 without consideration of fault models. k_B and $k_{\neg B}$ are modelled with (C, B) = [1 1] and (C, B) = [0 0], if e.g. the component R3 is present and $/k_B$ with (C, B) = [0 1], e.g. if R3 << R2. The coding of the functions k_C and $k_{\neg C}$ is already covered by $spec_{/D}$ and $spec_{\neg/D}$ with (/D, C, /D) = [0 1 0] and (/D, C, /D) = [1 0 1]. These are the edges from C to /D and ¬C to ¬/D in the corresponding SFG to Table 5. However, k_F and $k_{\neg F}$ are explicitly modelled in Table 8 and are holding the output state F. For the coding of remaining functions analogous procedure were applied. The encodings carried out can then be compared with the complete value tables (Table 2 in previous section). All encodings are contained in them and with that, everything is right.

Table 3. OP encoded in B and associated SFG

B	/A	S1	/S2	B	Function
1	0	*	*	1	**$spec_B$**
0	1	*	*	0	**$spec_{\neg B}$**
1	1	-	-	0	**$test_B$**
1	0	1	0	0	
0	0	1	1	1	**$test_{\neg B}$**
0	0	0	-	1	

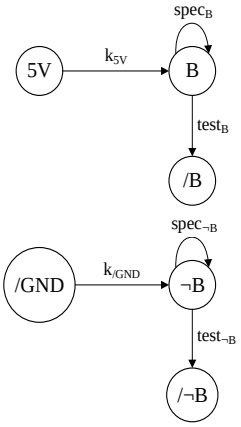


Table 4. (OP, CTRL) encoded in (C, B) and associated SFG

C	S1	/S2	C	Function
1	*	*	1	**$spec_C$**
0	*	*	0	**$spec_{\neg C}$**
1	-	0	0	**$test_C$**
1	0	-	0	

C	S1	/S2	B	Function
1	*	*	1	**k_B**
0	*	*	1	**$/k_B$**
0	*	*	0	**$k_{\neg B}$**

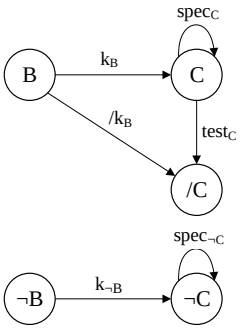


Table 5. OP encoded in /D and associated SFG

/D	S3	/S4	C	/D	Function
0	*	*	1	0	**$spec_{/D}$**
1	*	*	0	1	**$spec_{\neg/D}$**
1	1	0	*	0	**$test_{\neg/D}$**

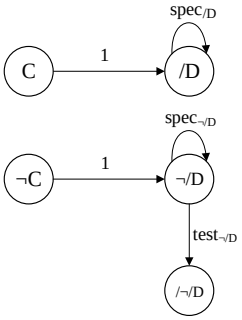


Table 6. (OP, CTRL) encoded in (/E, /D) and associated SFG

/E	S3	/S4	/E	Function
0	*	*	0	**$spec_{/E}$**
1	*	*	1	**$spec_{\neg/E}$**
1	-	0	0	**$test_{\neg/E}$**
1	0	-	0	

/E	S3	/S4	/D	Function
0	*	*	0	**$k_{/D}$**
1	*	*	0	**$/k_{/D}$**
1	*	*	1	**$k_{\neg/D}$**

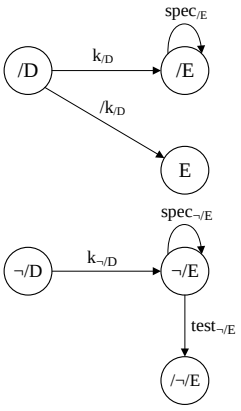


Table 7. OP encoded in F and associated SFG

F	/S5	S6	/E	F	Function
1	*	*	0	1	**$spec_F$**
0	*	*	1	0	**$spec_{\neg F}$**
1	0	-	*	0	**$test_F$**

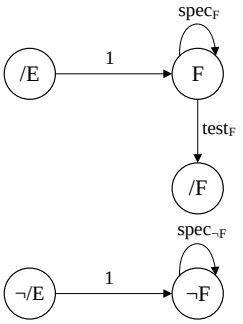
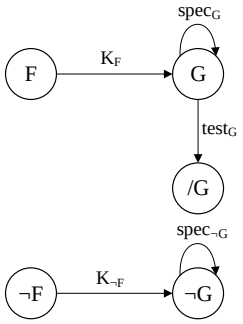


Table 8. (OP, CTRL) encoded in (G, F) and associated SFG

G	/S5	S6	G	Function
1	*	*	1	**$spec_G$**
0	*	*	0	**$spec_{\neg G}$**
1	-	0	0	**$test_G$**
1	0	-	0	

F	/S5	S6	F	Function
1	*	*	1	**k_F**
0	*	*	0	**$k_{\neg F}$**



Not all functions or state transitions are encoded in the SFG in Figure 3, $/k_C$, $/k_{/E}$, $/k_F$, $/k_{\neg B}$, $/k_{\neg C}$, $/k_{\neg/D}$, $/k_{\neg/E}$ and $/k_{\neg F}$ do not lead to any additional information, $test_{/D}$, $test_{/E}$, $test_{\neg C}$, $test_{\neg F}$ and $test_{\neg G}$ will never be fulfilled. Therefore, the states D, /¬C, /¬F and /¬G in Figure 3 are not achieved The merging of the SFGs corresponding to Table 3 to Table 8 are embedded into the SFG in Figure 4.

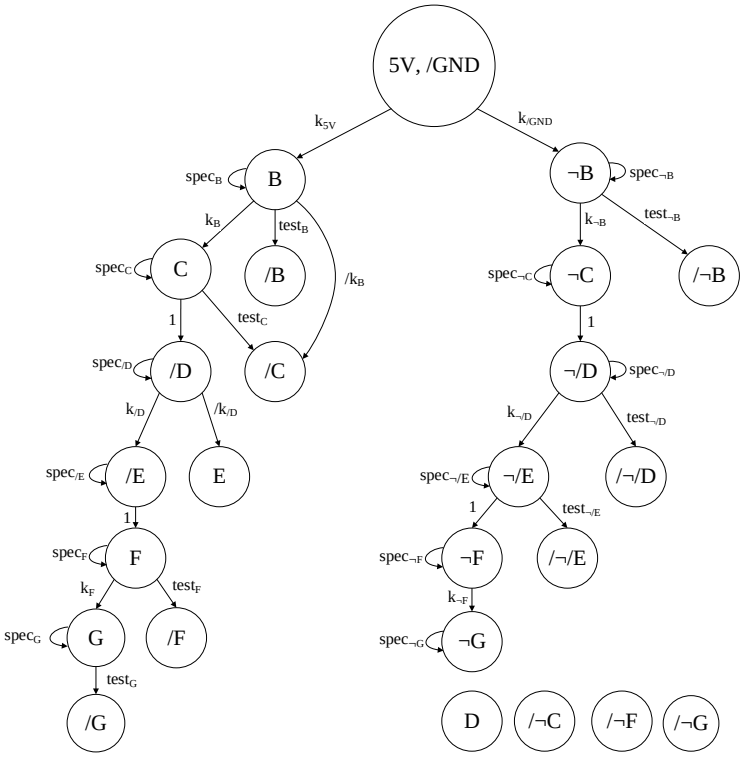


Figure 4. Data model (SFG) of DUT

3.2.3.3 Setting up of TVL in QVL

The TVLs created in section 3.2.3.2 are now to be set up and summarized as QVL. This is done by replacing the symbol "∗" in Table 3 to Table 8.The symbol assigns "undefined" with the symbol "X". Table 9 shows the summary of the TVL in QVL. QVL is a format for the realization of TVL. The function $\delta(Z^aZ^n)$ is constructed analogously to Table 1, the input and the switch are encoding δ. In the next section we briefly present a method to locate errors based on the results of our structure-preserving modelling method (data model Table 9).

Table 9. Data model of DUT (in QVL)

S^a						PI	Switch						S^n						Function
B	**C**	**/D**	**/E**	**F**	**G**	**/A**	**S1**	**/S2**	**S3**	**/S4**	**/S5**	**S6**	**B**	**C**	**/D**	**/E**	**F**	**G**	
1	X	X	X	X	X	0	X	X	X	X	X	X	1	X	X	X	X	X	$\mathbf{spec_B}$
0	X	X	X	X	X	1	X	X	X	X	X	X	0	X	X	X	X	X	$\mathbf{spec_{\neg B}}$
1	X	X	X	X	X	0	1	0	X	X	X	X	0	X	X	X	X	X	$\mathbf{test_B}$
1	X	X	X	X	X	1	-	-	X	X	X	X	0	X	X	X	X	X	
0	X	X	X	X	X	0	0	-	X	X	X	X	1	X	X	X	X	X	$\mathbf{test_{\neg B}}$
0	X	X	X	X	X	0	1	1	X	X	X	X	1	X	X	X	X	X	
X	1	X	X	X	X	X	X	X	X	X	X	X	1	X	X	X	X	X	$\mathbf{k_B}$
X	0	X	X	X	X	X	X	X	X	X	X	X	1	X	X	X	X	X	$\mathbf{/k_B}$
X	0	X	X	X	X	X	X	X	X	X	X	X	0	X	X	X	X	X	$\mathbf{k_{\neg B}}$
X	1	X	X	X	X	X	X	X	X	X	X	X	X	1	X	X	X	X	$\mathbf{spec_C}$
X	0	X	X	X	X	X	X	X	X	X	X	X	X	0	X	X	X	X	$\mathbf{spec_{\neg C}}$
X	1	X	X	X	X	X	0	-	X	X	X	X	X	0	X	X	X	X	$\mathbf{test_C}$
X	1	X	X	X	X	X	-	0	X	X	X	X	X	0	X	X	X	X	
X	X	0	X	X	X	X	X	X	X	X	X	X	X	1	0	X	X	X	$\mathbf{spec_{/D}}$
X	X	1	X	X	X	X	X	X	X	X	X	X	X	0	1	X	X	X	$\mathbf{spec_{\neg/D}}$
X	X	1	X	X	X	X	X	X	1	0	X	X	X	X	0	X	X	X	$\mathbf{test_{\neg/D}}$
X	X	X	0	X	X	X	X	X	X	X	X	X	X	X	0	X	X	X	$\mathbf{k_{/D}}$
X	X	X	1	X	X	X	X	X	X	X	X	X	X	X	0	X	X	X	$\mathbf{/k_{/D}}$
X	X	X	1	X	X	X	X	X	X	X	X	X	X	X	1	X	X	X	$\mathbf{k_{\neg/D}}$
X	X	X	0	X	X	X	X	X	X	X	X	X	X	X	X	0	X	X	$\mathbf{spec_{/E}}$
X	X	X	1	X	X	X	X	X	X	X	X	X	X	X	X	1	X	X	$\mathbf{spec_{\neg/E}}$
X	X	X	1	X	X	X	X	X	0	-	X	X	X	X	X	0	X	X	$\mathbf{test_{\neg/E}}$
X	X	X	1	X	X	X	X	X	-	0	X	X	X	X	X	0	X	X	
X	X	X	X	1	X	X	X	X	X	X	X	X	X	X	X	0	1	X	$\mathbf{spec_F}$
X	X	X	X	0	X	X	X	X	X	X	X	X	X	X	X	1	0	X	$\mathbf{spec_{\neg F}}$
X	X	X	X	1	X	X	X	X	X	X	0	-	X	X	X	X	0	X	$\mathbf{test_F}$
X	X	X	X	1	X	X	X	X	X	X	X	X	X	X	X	X	1	X	$\mathbf{k_F}$
X	X	X	X	0	X	X	X	X	X	X	X	X	X	X	X	X	0	X	$\mathbf{k_{\neg F}}$
X	X	X	X	X	1	X	X	X	X	X	X	X	X	X	X	X	X	1	$\mathbf{spec_G}$
X	X	X	X	X	0	X	X	X	X	X	X	X	X	X	X	X	X	0	$\mathbf{spec_{\neg/G}}$
X	X	X	X	X	1	X	X	X	X	X	0	-	X	X	X	X	X	0	$\mathbf{test_G}$
X	X	X	X	X	1	X	X	X	X	X	-	0	X	X	X	X	X	0	

3.3. ERROR LOCALIZATION

Fault diagnosis of complex systems gains more and more attention in the matter of system safety. The goal is to detect and isolate all faults of the system to ensure lowest costs and the reliability [10] [11]. We are now shortly introducing our method of fault diagnosis. After coding the functions spec, test, k and /k, we can now determine all scenarios for error localization. This is done using the function test. The data model for the real digital circuit (Figure 2) obtained here provides seven test functions: $test_B$, $test_{\neg B}$, $test_C$, $test_{\neg/D}$, $test_{\neg/E}$, $test_F$ and $test_G$. However, they are not all effective for detecting the error locations. There are test functions, detecting known errors

as being present, such as $test_B$, $test_{\neg B}$ in Table 3, which is important, but does not lead to error localization. There are these test functions, with which we can never identify whether an error exists, e.g. $test_{\neg B}$ in Table 3. $test_{\neg B}$ is fulfilled for the coding (B, /A, S1, /S2, B) = [0 0 0 - 1], so pin B (Z^n) receives signal 1 when /A = 0, regardless of which assignments the jumpers (error location) S1 and /S2. The same applies to $test_B$ with the coding (B, /A, S1, /S2, B) = [1 1 - - 0]. Table 10 describes all scenarios for error localization, which results of Table 3 to Table 8.

Table 10.Error localization

	$test_B$ = 1	**$test_c$ = 1**	**$test_{\neg/D}$ = 1**	**$test_{\neg/E}$ = 1**	**$test_F$ = 1**	**$test_G$ = 1**
S1	1	0	*	*	*	*
/S2	0	0	*	*	*	*
S3	*	*	1	0	*	*
/S4	*	*	0	0	*	*
/S5	*	*	*	*	0	0
S6	*	*	*	*	0	0

We call Table 10 a test-cover-table test(S), which shows us the covering of the errors by each individual test. As an example, $test_B$ is covering up only jumper S2 (/S2 = 0), while $test_C$ covers up S1 and/or S2 (S1, /S2) = [0 0]. The green marked positions are no errors, are thus usually no part of a test-cover-table, but give more information for error localization. These positions must be fulfilled, so that the corresponding test function is fulfilled (fail) e.g. $test_B$ = 1 when (S1, /S2) = [1 0]. Same applies to $test_{\neg/D}$. If test function $test_B$ shows fail ($test_B$ = 1), then jumpers S1 and S2 are closed. However, the error location is at the jumper, which shows code 0. Therefore, the fault is for sure jumper S2 (short circuit). The same applies to $test_{\neg/D}$ with error location jumper S4 (/S4 = 0). Is $test_C$ fail, then either S1 is open (electrical interruptions) and/or S2 is closed (short circuit), i.e. (S1 = 0) ∨ (/S2 = 0) ∨ (S1 = 0 ∧ /S2 = 0). The same applies to $test_{\neg/E}$ and $test_G$ with respective jumper combination (error locations) (S3, S4) and (S5, S6). In the cases $test_C$, $test_{\neg/E}$ and $test_G$, there are two possibilities each for error location. Function $test_F$ is fulfilled when either jumper S5 is open or S5 is open and S6 is closed, i.e. (/S5 = 0) ∨ (/S5 = 0 ∧ S6 = 0), coloured grey for illustration purposes. In this case, it can be assumed with certainty that if $test_F$ is fail, then S5 is an error location, but S6 could be an additional error location, too. If $test_B$ and $test_C$ is fail, then for sure jumper S2 is the error location (/S2 = 0, short circuit), because of the coding of jumper S1 in both $test_B$ and $test_C$. Same applies for $test_{\neg/D}$ and $test_{\neg/E}$ for jumper S4 (/S4 = 0). The data model is total in terms of test function, since it provides all possible test functions (effective and non-effective) and thus contains all possible error locations. This analysis, based on the test-cover-table, can be illustrated with the aid of SFG in a more detailed and understandable way. Following SFG (Figure 5) shows a fault tree. This is a 1:1 projection of Table 10 not including the green marked positions. It serves as a possible method to quickly identify involved defective components, in order to then exchange them specifically. This fault tree consists of six paths, each covering 100% of fail tests and each representing a different exchange possibility. When analyzing the fault tree, it does not have to be run through completely, as we jump to the next node accordingly to fail and pass of tests, thus changing the component involved. The replacement of the components and the execution of fail test happen iteratively, without repetition of the exchange procedures, until all tests result in pass. If, despite the run of all paths in the entire fault tree, not all tests result in pass, this means that our data model (Table 9) is incomplete (absence of additional information) and thus the fault tree confirms itself. It has a direction from top to bottom and from left to right, depending on the rank of the components involved and therefore this method is structure-preserving, too. In order to better comprehend the application of the structure-preserving modelling method at the DUT, and to confirm the results of the method for Error localization, we will present in the next section four use cases as scenarios for DUT testing. The visualization is done by representation of SFGs.

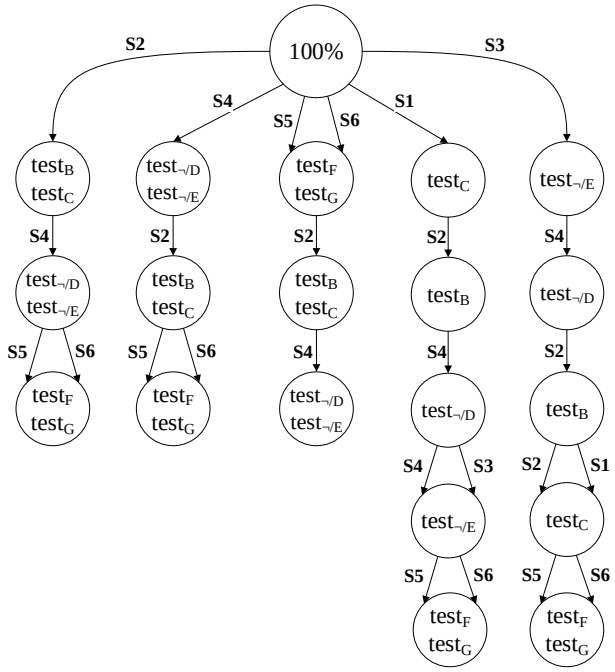


Figure 5. Fault tree of DUT

3.4. USE CASES

In this section, the expected results of the step-by-step verification method for the predefined digital circuit are now presented. Through a self-written program in VBA (Visual Basic for Applications) is-values shall be compared to set-values from Table 9. The program visualizes the correspond-ing SFG, coming from the resulting data model from Figure 4 with the exception of four not reached states D, /¬C, /¬F and /¬G. All phase lists (edges with their nodes) in this SFG are without restriction of the generality as adjoining. The functions spec, test, k and /k are coloured, if they are fulfilled, as green, red, black and blue arrows. If they are not, dashed arrows serve as their visualization. In the first case (Figure 6) all switches (jumpers) operate in normal mode (the known faults are not present), so obtaining a digital 1 in the is-table. Input /A has a digital 0. Since spec is fulfilled regarding B, C, /D, /E, F and G, each state (B, C, /D, /E, F, G) maintains its value ($S^n = S^a$) and the corresponding edges are represented as green arrows. Similarly, k regarding /B, /D and F are fulfilled. The remaining functions are not fulfilled due to the different is- and set-values and are marked accordingly. In the second case in Figure 7, jumper S2 is set to closed, receiving consequently the digital 0 (/S2 = 0). All other is-values are same as in case 1. Here, spec is no longer fulfilled regarding all pins, whereas $test_B$ and $test_C$ are fulfilled. We gained this information already before from Table 10 and corresponding SFG in Figure 5. The is-table for the third case is filled similar to the first case, so that now spec is fulfilled in the dual rail regarding ¬B, ¬C, ¬/D, ¬/E, ¬F and ¬G. In the last case (Figure 9), S4 is set to closed, so it receives the digital 0 (/S4 = 0), the other is-values stay the same as in Figure 8. Hence, spec is no longer fulfilled regarding ¬/D, ¬/E and /F but test regarding ¬/D and ¬/E are fulfilled (Table 10 and Figure 5). In this case, it is still to mention, that /k regarding /D is fulfilled because of the accordance of its is- and set-values and marked accordingly (blue). Similarly, other examples can be generated. The examples presented show how to take care of analog properties like voltage and current, which are known as parametric measures. Indeed, the methodology is not limited to those numbers. It can additionally be applied to other physical constraints as well as all kind of event based digital forms. To be mentioned are delay, energy and power. The effort there will also be in the modelling of the underlying structure in an abstraction of a SFG. Now, search functions can be programmed which go through the QVL (Table 9), determine certain criteria, e.g. test coverage, fault coverage and test severity. However, the QVL generated by a complex

reality can be of great dimensions. Therefore, the database (QVL) should be compressed without information loss (data compaction) in order to ensure lower memory requirements and shorter computing time. The next chapter illustrates the procedure for such a lossless data compression.

S^a						PI	Switch						S^n					
B	C	/D	/E	F	G	/A	S1	/S2	S3	/S4	/S5	S6	B	C	/D	/E	F	G
1	1	0	0	1	1	0	1	1	1	1	1	1	1	1	0	0	1	1

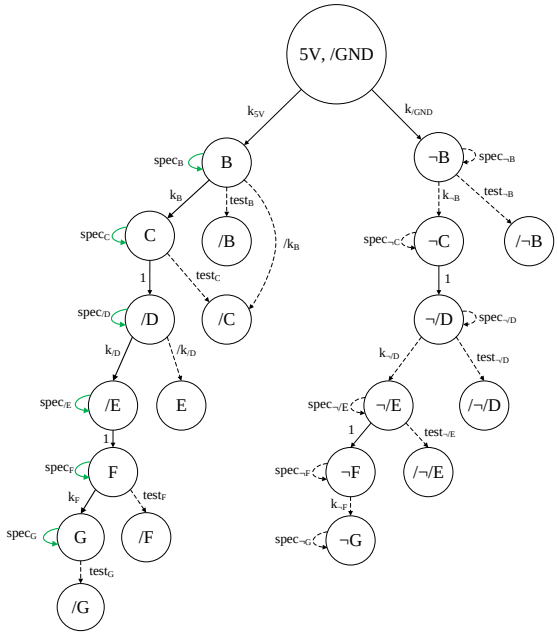


Figure 6. Case 1: Is-table and SFG

S^a						PI	Switch						S^n					
B	C	/D	/E	F	G	/A	S1	/S2	S3	/S4	/S5	S6	B	C	/D	/E	F	G
1	1	0	0	1	1	0	1	0	1	1	1	1	0	0	1	1	0	0

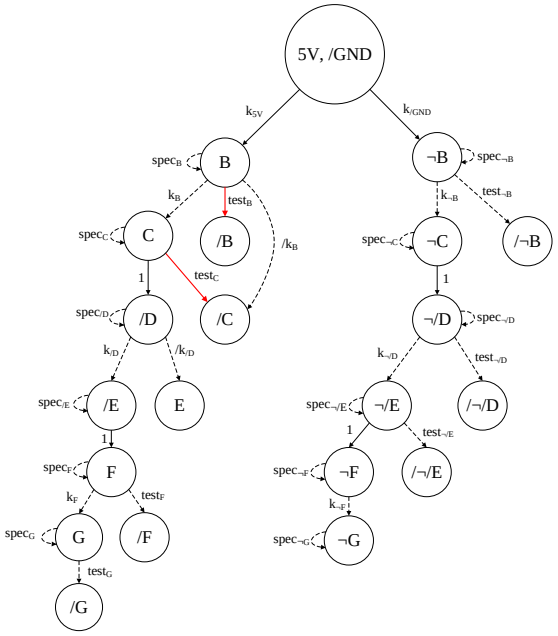


Figure 7. Case 2: Is-table and SFG

S^a						PI	Switch						S^n					
B	C	/D	/E	F	G	/A	S1	/S2	S3	/S4	/S5	S6	B	C	/D	/E	F	G
0	0	1	1	0	0	1	1	1	1	1	1	1	0	0	1	1	0	0

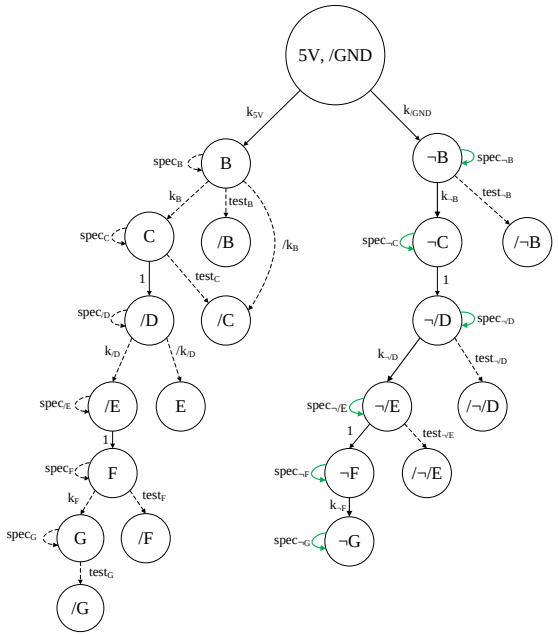


Figure 8. Case 3: Is-table and SFG

S^a						PI	Switch						S^n					
B	C	/D	/E	F	G	/A	S1	/S2	S3	/S4	/S5	S6	B	C	/D	/E	F	G
0	0	1	1	0	0	1	1	1	1	0	1	1	0	0	0	0	1	1

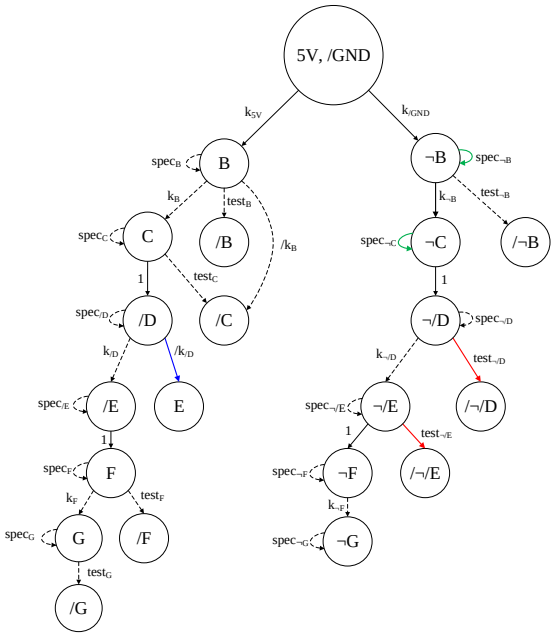


Figure 9. Case 4: Is-table and SFG

4. DATA COMPACTION

4.1. THEORY

Table 11. Table of operations

¦	**0**	**1**	∗	-
0	0	id	$*_0$	$-_0$
1	id	1	$*_1$	$-_1$
∗	0_*	1_*	∗	$-_*$
-	0_-	1_-	$*_-$	-

Table 11 as a table of operations represents the position-by-position composition of QVL. The symbol "¦" represents, according to our chosen definition, the positional, regular compaction of two quaternary vectors QV1 and QV2. In Table 11, the first column refers to QV1 and the first row to QV2. The diagonal is self-explanatory, e.g. ¦ (0,0) = 0. The remaining points are filled with symbols, which are considered to be intermediate results of the compaction. Symbol "id" is assigned to the regular compaction of (0,1) with (1,0). The symbol "$*_0$" means that QV2 as "∗" is a subset of QV1 as "0", the star from QV2 is concretized in the "0" of QV1. Symbol "$-_0$" then means that QV2 as "-" is defined as superset of QV1 as "0", the "-" of QV2 is concretized in "0" of QV1. Other symbols are valid accordingly. In order to achieve a completely parallel processing during compaction, the sequential quaternary list coding (intermediate results) from Table 11 is additionally embedded in a five-digit coding universe (columns 2 to 6 of Table 12). Amount strokes accumulate the occurrence of the symbol indicated therein. For example, | id | = 0 means that "id" does not occur in the regular compaction of the two quaternary vectors QV1 and QV2, while | id | = 1 means that "id" occurs exactly once. | $(0,1,-)_*$ | > 0 means that "0_*", "1_*" and/or "$-_*$" occur at least once. The first five columns in Table 12 indicate all existing possible combinations in which a compaction exists. The remaining five columns then indicate the symbols with which symbols (intermediate results of data compaction) from Table 11 must be replaced. The more concrete TVLs can also be compacted with the presented method, these are the lines 1 to 3 and 9 to 12 for | $(0,1,-)_*$ | = | $*_{(0,1,-)}$ | = 0. There is no compacting for all other combinations (lines), not listed in the first five columns in Table 12. There is a subset relation in the first eight lines of Table 12, in line 9 a consensus and in the last three lines an overlap. If the coding of the intermediate result corresponds to one of the combinations of the first nine lines of Table 12, then the two quaternary vectors are subsets of the final result and are therefore replaced by the final result. If the encoding matches line 10, the end result will replace QV1, because here QV2 is overlapping QV1 and thus QV1 is a subset of the final result.

Table 12. All Existent combinations and their replacement values

row	**| id |**	**$\|(0,1,-)_*\|$**	**$\|*_{(0,1,-)}\|$**	**$\|(0,1)_-\|$**	**$\|-_{(0,1)}\|$**	**id**	**$(0,1,-)_*$**	**$*_{(0,1,-)}$**	**$(0,1)_-$**	**$-_{(0,1)}$**
1	0	0	0	0	0	∗	∗	∗	∗	∗
2	0	0	0	0	> 0	∗	∗	∗	∗	-
3	0	0	0	> 0	0	∗	∗	∗	-	∗
4	0	0	> 0	0	0	∗	∗	(0,1,-)	∗	∗
5	0	0	> 0	> 0	0	∗	∗	(0,1,-)	-	∗
6	0	> 0	0	0	0	∗	(0,1,-)	∗	∗	∗
7	0	> 0	0	0	> 0	∗	(0,1,-)	∗	∗	-
8	0	> 0	> 0	0	0	∗	(0,1,-)	(0,1,-)	∗	∗
9	1	0	0	0	0	-	∗	∗	∗	∗
10	1	0	0	0	>0	-	∗	∗	∗	(0,1)
11	1	0	0	>0	0	-	∗	∗	(0,1)	∗
12	1	0	0	>0	>0	-	∗	∗	(0,1)	(0,1)

The same applies to QV2 as a subset of the final result (line 11). If both quaternary vectors are not subset of the final result, the end result acts as a further quaternary vector to serve a progressive compaction. This is in line 12 the case, because "-" is in QV1 as well as in QV2 in different places. As a result, the QVL becomes larger as in the case of the Quine-McCluscy method [12], at least becoming smaller as the compaction progresses.

Table 13. Calculation examples for data compaction

case 1	**case 2**	**case 3**	**case 4**	**case 5**	**case 6**
0 1 0	0 1 - 1	∗ 1 0	0 ∗ 1 ∗	1 0 1	1 1 = 1 1 + - 1 + - -
¦ - 1 0	¦ ∗ 1 0 ∗	¦ 1 - 0	¦ ∗ - ∗ 1	¦ 0 0 ∗	0 - 0 - - 0 - 0 -
$-_0$ 1 0	$*_0$ 1 0 $-_*$ $*_1$	1_* $-_1$ 0	$*_0$ $-_*$ $*_1$ 1_*	id 0 $*_1$	- 0 id - 1 - id 0 -
⇒ **- 1 0**	⇒ **0 1 - 1**	⇒ **1 - 0**	⇒ **0 - 1 1**	⇒ **∗ ∗ ∗**	- - = - 1 = - - = - -

With the help of the calculation examples from Table 13, it is easy to comprehend: In case 1, the intermediate result of the regular compaction follows [0 1 0] ¦ [- 1 0] = [$-_0$ 1 0]. The coding of this intermediate result, [0 0 0 0 >0], corresponds to the second line in Table 12, which leads to "$-_0$" being replaced by "-". Cases 2 to 4 are to be understood analogously and refer to lines 5, 7 and 8 from Table 12. The intermediate result in case 5 corresponds to an encoding of [1 0 >0 0 0], which does not occur in any of the lines from Table 12. Thus the intermediate result [id 0 $*_1$] is not used any further. In case 6, three ternary-vectors (TVs) are considered and coloured for better understanding. First, the final result of the compacting of TV1 (blue) and TV2 (red) is [-1], which follows from line 10 in Table 12. This result covers TV1, which causes TV1 to be replaced with [- 1] and subsequently compacted with TV3 (green). According to line 9 from Table 12, this results in the substitution of both ternary vectors with [- -]. This result is compacted with the still remaining TV2 and from line 3 follows the replacement of both TVs with the final result [- -].

4.2. REALIZATION

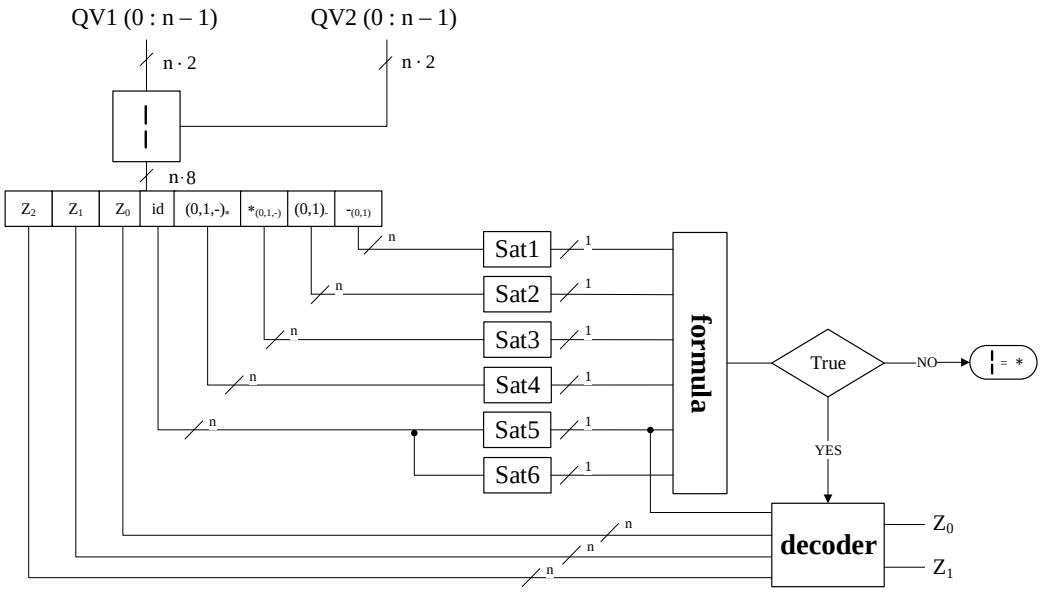


Figure 10. Block diagram of the data compaction method

Figure 10 illustrates a block diagram as circuit for realizing the explained theory of data compaction (section 4.1). This leads to the compaction of two QVLs, if possible. Two QVL (QV1 and QV2) of length n (n ∈ N) are available as input. QV1 and QV2 each have n times 2 entries (conduits) since the symbols (0, 1, -, ∗) are divalent (binary) coded (Table 14). QV1 and QV2 are

then entered as input in operation "¦" in the first block. The table of operations in Table 11 can be decomposed, after closer examination of Table 12, into two individual tables of operations (Table 15). The intermediate results for each | id | = 0 and | id | = 1 are thus fix. This can be understood more precisely by considering the intermediate results in the last five columns in Table 12.

Table 14. Coding of the symbols

Symbol	Code
"**0**"	01
"**1**"	10
"**-**"	11
"*****"	00

Table 15. Table of operations

| id | = 0

¦	**0**	**1**	*****	**-**
0	0	-	0	-
1	-	1	1	-
*****	0	1	*	-
-	-	-	-	-

| id | = 1

¦	**0**	**1**	*****	**-**
0	0	-	0	0
1	-	1	1	1
*****	0	1	*	-
-	0	1	-	-

In the first eight lines in Table 12, with the commonality of | id | = 0, the intermediate results $-_{(0,1)}$ as well as (0,1)., when existing, were replaced by "-" ([11]), whereas in the last four lines (| id | = 1) replacement was carried out with "(0,1)" ([01], [10]). The two resulting tables in Table 15 differ in only four out of 16 digits (coloured grey). For the first workflow "¦", we first decide for the table of operations to | id | = 1, with a possible correction at the end if necessary, i.e. when | id | = 0 appears.

Table 16. QV1 compaction with QV2 for | id | = 1

row	QV1	QV1	Z_2	Z_1	Z_0	id	$(0,1,-)_*$	$*_{(0,1,-)}$	$(0,1)_-$	$-_{(0,1)}$
1	01	01	0	0	1	0	0	0	0	0
2	00	01	0	0	1	0	1	0	0	0
3	01	00	0	0	1	0	0	1	0	0
4	10	10	0	1	0	0	0	0	0	0
5	00	10	0	1	0	0	1	0	0	0
6	10	00	0	1	0	0	0	1	0	0
7	00	00	0	0	0	0	0	0	0	0
8	10	01	0	1	1	1	0	0	0	0
9	01	10	0	1	1	1	0	0	0	0
10	11	11	0	1	1	0	0	0	0	0
11	00	11	0	1	1	0	1	0	0	0
12	11	00	0	1	1	0	0	1	0	0
13	01	11	1	0	1	0	0	0	0	1
14	11	01	1	0	1	0	0	0	1	0
15	10	11	1	1	0	0	0	0	0	1
16	11	10	1	1	0	0	0	0	1	0

All possible input combinations of (QV1, QV2) (2^4 = 16 combinations) are given in Table 16. As output we get a list of length n with eight entries (Z_2, Z_1, Z_0, id, $(0,1,-)_*$, $*_{(0,1,-)}$, $(0,1)_-$, $-_{(0,1)}$). They are outputted independently and in parallel. The variables Z_1 and Z_0 are used for the binary coding of the results from Table 15 to | id | = 1. Auxiliary variable Z_2 provides additional and necessary information as to whether we are in the grey area of Table 15. This is encoded with Z_2 = 1 (grey) and Z_2 = 0 (not grey). The remaining symbols (id, $(0,1,-)_*$, $*_{(0,1,-)}$, $(0,1)_-$, $-_{(0,1)}$) correspond to the same symbols from the last five columns in Table 12. They are encoded with 1 if they exist as intermediate results for a certain combination of (QV1, QV2), otherwise with 0. The last 5 times n outputs of operation "¦" (id, $(0,1,-)_*$, $*_{(0,1,-)}$, $(0,1)_-$, $-_{(0,1)}$) are first entered parallel into six satellites (Sat1 to Sat6). The satellites serve as functions for detecting the presence of each of these intermediate results and operate in parallel and independently. For the first five satellites, a single value is obtained for each satellite: 1 (True) if the investigated intermediate results occur at least once, or 0 (False) if they do not occur at all. However, Sat6 checks whether

the intermediate result "id" occurs exactly once. All satellites are initialized to True. However, as soon as one of the satellites Sat1 to Sat5 receives the signal 1, i.e. the intermediate result is existing, the run is terminated (while-loop) and the corresponding satellite gives False as output. This is the principle of Wired-OR, so it is always checked for signal 0. This completes the calculation quickly. The realization of the first five satellites Sat1 to Sat5 can be seen as a flow chart with the corresponding program code in Figure 11 und Figure 12. Sat6 needs an additional auxiliary variable, which we call flag, to investigate whether "id" occurs exactly once. Sat6 is initially initialized to False (| id | = 0). Once 1 is signalled, Sat6 takes the value of flag, which has been initialized True and flag becomes False. If an additional 1 is signalled, Sat6 takes the value of Flag with False and thus | id | > 1. When no additional 1 signal is received, the value remains True and thus | id | = 1. The flow chart and program code associated with Sat6 are shown in Figure 13 and Figure 14. The input variable tv in Figure 11, Figure 12, Figure 13 and Figure 14 represents the outputs id, $(0,1,-)_{*}$, $*_{(0,1,-)}$, $(0,1)_{-}$, and $-_{(0,1)}$ of the operation "¦" from Figure 10. They are TVLs of length n and are examined in descending order starting from tv(n-1). Since all satellites work in parallel and independently, the first five satellites need only one cycle, whereas Sat6 requires two cycles. The monovalent outputs of the six satellites (True or False, i.e. 1 or 0) flow into a formula, explaining the first five columns in Table 12.This formula is used to decide whether the results Z_1 and Z_0 of the operation "¦" will be further accepted and whether QV1 and QV2 can be compacted. In order to enable a faster decision the following steps were made: Firstly, the first five columns in Table 12 were reduced line-by-line, i.e. lossless compressed and secondly, the columns where ordered in a more intelligent way. This results in Table 17.

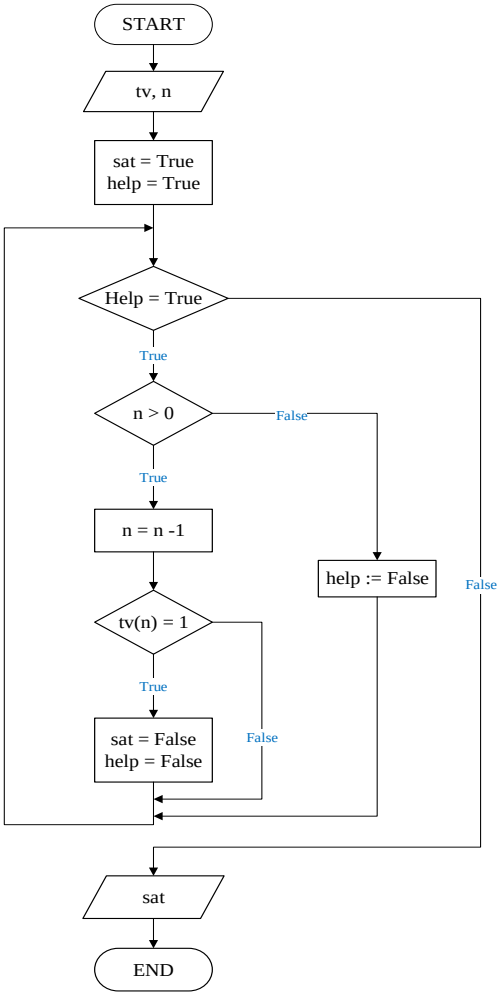


Figure 11. Flow chart of Sat1 to Sat5

```
Algorithm | tv | = 0
Input: tv, n
Output: sat
sat = True
help = True
While help = True
    if n > 0 then
        n = n - 1
        if tv(n) = 1 then
            sat = False
            help = False
        end if
    else
        help = False
    end if
Wend
return sat
```

Figure 12. Algorithm of Sat1 to Sat5

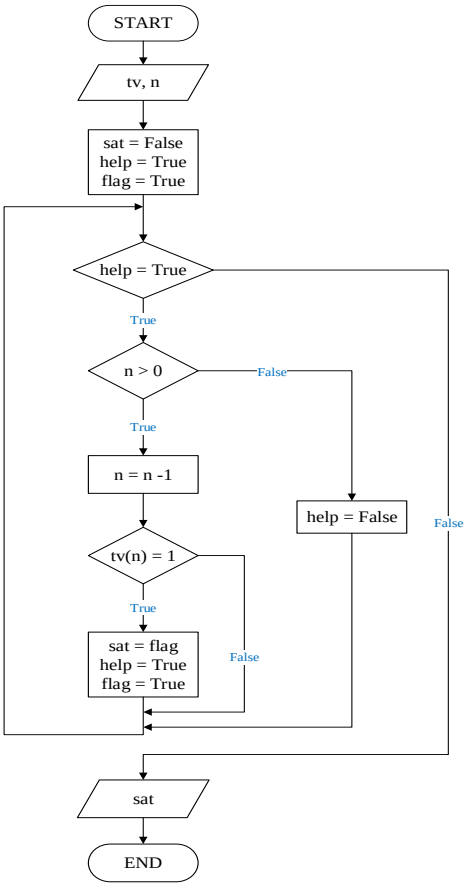


Figure 13. Flow chart of Sat6

```
Algorithm | tv | = 1
Input: tv, n
Output: sat
sat = False
help = True
flag = True
While help = True
    if n > 0 then
        n = n - 1
        if tv(n) = 1 then
            sat = flag
            help = flag
            flag = False
        end if
    else
        help = False
    end if
Wend
return sat
```

Figure 14. Algorithm of Sat6

Table 17. All existent combinations (compacted)

row	| id |	$\| (0,1)_{-} \|$	$\| *_{(0,1,-)} \|$	$\| -_{(0,1)} \|$	$\| (0,1)_{*} \|$
1	0	0	0	-	-
2	0	0	> 0	0	-
3	0	> 0	-	0	0
4	1	-	0	-	0

The formula checks every possible combination of Table 17. If the combination occurs, then True results, i.e. the compaction can continue. In case of False, therefore, no compaction is possible. The corresponding flow chart and code for the formula are shown in Figure 15 and Figure 16. The Boolean variables sat_1 to sat_6 represent the outputs of satellites Sat1 to Sat6, sat_1 = True if $| -_{(0,1)} | = 0$, otherwise sat_1 = False. Same applies to sat_2, sat_3, sat_4 and sat_5 with each $| (0,1)_{-} |$, $| *_{(0,1,-)} |$, $| (0,1)_{*} |$ and | id |. For the Boolean variable sat_6 = True applies if | id | = 1, otherwise False. If we are in one of the four combinations in Table 17 (corresponding the 12 four combinations in Table 12), i.e. the output of the formula is True, then a black box, which we call decoder, gets the outputs from Z_2, Z_1, Z_0 (TVLs with n entries) and from Sat5 (True for | id | = 0, otherwise False). The decoder passes the results Z_1 and Z_0 when Sat5 yields False, i.e. | id | = 1, as before Table 15 to | id | = 1 was chosen. However, if Sat5 yields True, i.e. | id | = 0, the entries in Z_1 and Z_0 are each encoded with 1, if auxiliary variable Z_2, in the same locations, was encoded with 1 (grey region in Table 16). Table 18 and the associated program code (Figure 17) serve as explanation of the here made theory. For purposes of illustration, the places with Z_2 = 1 and | id | = 0 are marked in grey in Table 18.The output of the circuit (Figure 10) is then Z_1 and Z_0. The circuit realized here for data compaction works fully parallel in, which enables a fast compaction. The codes mentioned here are sequential as we work with a sequential computer and program language. However, thanks to the circuit in Figure 10, the compaction method presented here can be implemented fully in parallel in e.g. FPGA [13].

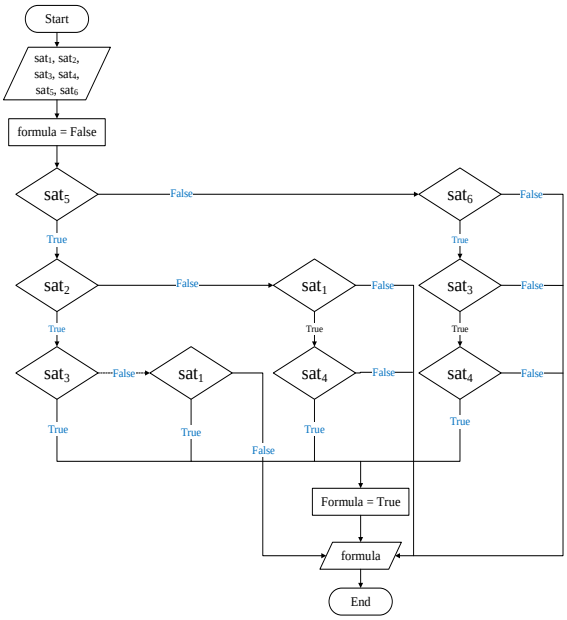


Figure 15. Flow chart of the formula

Algorithm formula
Input: sat_1, sat_2, sat_3, sat_4, sat_5, sat_6
Output: formula
Formula = False
if sat_5 = True **then**
  if sat_2 = True **then**
    if sat_3 = True **then**
      formula = True
    else
      if sat_1 = True **then**
        formula = True
      end if
    end if
  else
    if sat_1 = True **then**
      if sat_4 = True **then**
        formula = True
      end if
    end if
  end if
else if sat_6 = True **then**
  if sat_3 = True **then**
    if sat_4 = True **then**
      formula = True
    end if
  end if
end if
return formula

Figure 16. Algorithm of the formula

Table 18. Decoding of Z_1 and Z_2

Z_2	Z_1	Z_0	| id | = 0		| id | = 1	
			Z_1	Z_0	Z_1	Z_0
0	0	0	0	0	0	0
0	0	1	0	1	0	1
0	1	0	1	0	1	0
0	1	1	1	1	1	1
1	0	1	1	1	0	1
1	1	0	1	1	1	0

Algorithm decoder
Input: n, Z_2, Z_1, Z_0, sat_5
if sat_5 = True **then**
  for i = 0 **to** n-1
    if $Z_2(i)$ = 1 **then**
      $(Z_1(i),Z_0(i))$= [11]
    else
      $(Z_1(i),Z_0(i))$= $[Z_1(i),Z_0(i)]$
    end if
  next i
else
  (Z_1,Z_0) = $[Z_1,Z_0]$
return (Z_1,Z_0)

Figure 17. Algorithm of decoder

The result QV = (Z_1, Z_0) must now be assigned. In section 4.1 the possible replacement theories were mentioned: Depending on the correspondence of the coding of the intermediate results of calculation QV1 ¦ QV2 in Table 12, QV can substitute QV1, QV2 or both or can be used as new QV. This information of the replacement processes can be obtained with the help of the declared satellites: Sat1, Sat2 and Sat5 for each | $-_{(0,1)}$ | = 0, | (0,1)- | = 0 and | id | = 0 with their outputs sat_1, sat_2 and sat_5. Table 19 and the corresponding program code in Figure 18 serve as formula and are explaining the replacement processes. In the algorithm, the function "replace" serves as replacement of QV1 or/and QV2 while "addnew" provides a new place for QV.

Table 19. Overlapping possibilities of QVs

id	$(0,1,-)_*$	$*_{(0,1,-)}$	$(0,1)_-$	$-_{(0,1)}$	comment
0	0	0	0	0	**QV overlaps QV1 and QV2**
0	0	0	0	> 0	
0	0	0	> 0	0	
0	0	> 0	0	0	
0	0	> 0	> 0	0	
0	> 0	0	0	0	
0	> 0	0	0	> 0	
0	> 0	> 0	0	0	
1	0	0	0	0	
1	0	0	0	>0	**QV overlaps QV1**
1	0	0	>0	0	**QV overlaps QV2**
1	0	0	>0	>0	**No overlap**

Algorithm formula for the replacement

Input: sat_1, sat_2, sat_5, QV1, QV2, QV
Outputs: QV
if sat_5 = True **then**
    QV = replace (QV1,QV2)
else
    if sat_2 = True **then**
        if sat_1 = True **then**
            QV = replace (QV1,QV2)
        Else
            QV = replace (QV1)
        End if
    else
        if sat_1 = True **then**
            QV = replace (QV2)
        else
            QV = addnew(QV)
        end if
    end if
end if
return QV

Figure 18. Algorithm for replacement

The compaction procedure explained here is applied iteratively, with the aid of a self-compiled program code of the circuit (Figure 10), to the QVL of the achieved model in Table 9, which leads to Table 20. In Table 20, (s, t, n) = (spec, test, ¬). We have reduced the amount of 33 lines to seven without losing information. Here, e.g., the function $test_B$ (t_B) is fulfilled, i.e. is given the value 1, when for its encoding universe (B, /A, S1, /S2, B) = [1 0 1 0 0 ; 1 1 - - 0] applies (grey coloured for visualization). This condition corresponds to the same from Table 9. Analogously, all further functions (spec, test, k, /k) can be controlled and thus the data compaction can be confirmed. We achieved a lossless, fast and effective data compaction for QVL as well as for TVL. This saves the memory requirement as well as the computing time when searching for specific criteria or information. Thus, it generally reduces the expense and in our case, especially the test costs.

Table 20. Compacted data model of DUT (in QVL)

S^a						PI	Switch						S^n						Function
B	**C**	**/D**	**/E**	**F**	**G**	**/A**	**S1**	**/S2**	**S3**	**/S4**	**/S5**	**S6**	**B**	**C**	**/D**	**/E**	**F**	**G**	
1	0	1	1	0	1	0	X	X	X	X	X	X	1	0	1	1	0	1	s_B, $/k_B$, s_{nC}, $s_{n/D}$, $k_{n/D}$, $s_{n/E}$, s_{nF}, k_{nF}, s_G
0	0	0	0	1	0	1	X	X	X	X	X	X	0	1	0	0	1	0	s_{nB}, k_{nB}, $s_{/D}$, $k_{/D}$, $s_{/E}$, s_F, k_F, s_{nG}
1	1	1	1	1	1	0	1	0	1	0	0	-	0	1	0	X	0	0	t_B, s_C, $t_{n/D}$, $/_{k/D}$, t_F, t_G
1	X	X	1	X	1	1	-	-	0	-	-	0	0	X	X	0	X	0	t_B, $t_{n/E}$, t_G
0	1	X	1	X	X	0	0	-	-	0	X	X	1	0	X	0	X	X	t_{nB}, t_C, $t_{n/E}$
0	X	X	X	X	X	0	1	1	X	X	X	X	1	X	X	X	X	X	t_{nB}
X	1	X	X	X	X	X	-	0	X	X	X	X	1	0	X	X	X	X	k_B, t_C

5. CONCLUSIONS

The introduced Structure-Preserving Modelling based on SFG, is a structurally reliable verification method, which is used to test systems or circuits for known faults. By structure-preserving modelling, we mean the conformity of the formally derived function with the real generated function. The verification can be carried out, with the aid of certain derived rules, in three steps: embedding of the real model into a coding universe, specifying a data model as SFG in dual rail and the creation of the sub-models (OP, CTRL) in AA, which results in the establishment of a QVL. Compared to known methods such as simulation and validation, the proposed methodology takes care of the direction of the structure and can therefore preserve the structure of the device to be tested within its directional structure. In our terminology, this is named verification. Thus, it is possible to apply algebraic methods, which have been concluded under idempotency. A result of this is a faster and more compact code. However, to what extent such approaches can be taken, will be shown. The effort for the modelling of the underlying structure lies in the abstraction of the SFG. We then shortly presented a possible method, which analyses the results of our suggested verification method (data model), for error localization. The method bases on the coverage of the faults by tests (test-cover-table). This information was projected in a SFG, which we called fault tree. It serves as a possible method to quickly identify involved defective components, in order to then exchange them specifically, which is reducing enormous test costs. The method is easy to understand and suitable for complex system, which consist of many sub-systems and thus components. It preserves the direction and confirms itself, which makes the method a robust, structure-preserving method of fault diagnosis. Each assembly of a product contains a test coverage table, thus a fault tree. If all test coverage tables are superimposed, i.e. considered parallel, we get important additional information about the dependency between tests and the involved components. This can additionally be applied adaptively and thus a machine learning can be achieved [14] [15] [16]. In addition to the better understanding of complex systems and the interdependency between their components, it also provides fast, accurate and cost-effective error localization. The lossless compression (compaction) shown here ensures a small requirement or storage space and thus low computing time. It makes the test process compact and thus saves costs. QVL as well as TVL can be compacted. The realization of the data compaction in full parallel qualifies this method for an implementation in FPGA. This line-wise compacted database is column-wise multidimensional for its functional part, e.g. in dependencies or correlations between the individual functions (spec, test, k, / k) and their possible error models. Thus, it is useful to investigate such correlations also from the aspect of classification methods, which can reduce and classify the dimensions of data sets to relevant features. The Principal Component Analysis and the Linear Discriminant Analysis would enable this data classification [17].

6. ACKNOWLEDGEMENTS

We would cordially like to thank T. Egelhofer and B. Scheffold from Rohde & Schwarz for providing their real world expertise and fruitful discussion on error localization.

REFERENCES

[1] Xu. Y, Algorithms for Automatic Generation of Relative Timing Constraints, Ph.D. dissertation, Salt Lake City, UT, USA. (2011).

[2] Kurshan. R. P & McMillan, K. L, Analysis of digital circuits through symbolic reduction, IEEE Trans. on CAD of Integrated Circuits and Systems, pp. 1356–1371, (1991).

[3] F. Poehl & F. Demmerle & J. Alt & H. Obermeir, Production test challenges for highly integrated mobile phone SOCs - A case study, 15th IEEE European Test Symposium, Praha, pp. 17-22, (2010).

[4] Yoneda. T & Kitai. T & Myers. C. J, Automatic Derivation of Timing Constraints by Failure Analysis in CAV, vol. 2404. Springer, pp. 195–208. (2002).

[5] Uygur. G & Sattler. S, Structure Preserving Modeling for Safety Critical Systems, IEEE 20th International Mixed-Signals Testing Workshop (IMSTW), (2015).

[6] S. Wendt, Operation state versus control state (Operationszustand versus Steuerzustand - eine äußerst zweckmäßig Unterscheidung -), Technical University Kaiserslautern, (2000).

[7] Posthoff. Ch & Steinbach. B, Logic Functions and Equations, Springer, (2004).

[8] Gössel. M, Automata theory for engineers, Academy published, Berlin, (1991).

[9] Zander. H.J, Logical design of binary systems (german: Logischer Entwurf binärer Systeme), VEB Technology published . Berlin, (1989).

[10] F. Basile, Overview of fault diagnosis methods based on Petri net models, 2014 European Control Conference (ECC), Strasbourg, pp. 2636-2642, (2014).

[11] L. Zhang, Research on fault diagnosis test sequence algorithm based on multi-signal flow graph model, 2017 Second International Conference on Reliability Systems Engineering (ICRSE), Beijing, pp. 1-7, (2017).

[12] E. McCluskey, Logic Design Principles, Prentice-Hal, Englewood Cliffs, New Jersey, (1986).

[13] B. Bond & K. Hammil & L. Litchev & S. Singh, FPGA Circuit Synthesis of Accelerator Data-Parallel Programs, 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, Charlotte, NC, pp. 167-170, (2010).

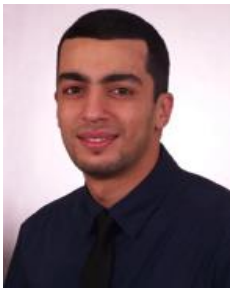
[14] M. J. Barragan & G. Leger, Efficient selection of signatures for analog/RF alternate test, 18th IEEE European Test Symposium (ETS), Avignon, pp. 1-6, (2013).

[15] E. Yilmaz & S. Ozev & K. M. Butler, Adaptive multidimensional outlier analysis for analog and mixed signal circuits, IEEE International Test Conference, Anaheim, CA, pp. 1-8, (2011).

[16] H. G. Stratigopoulos & P. Drineas & M. Slamani & Y. Makris, RF Specification Test Compaction Using Learning Machines, in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 18, no. 6, pp. 998-1002, (2010).

[17] Xie. Y & Zhang. T, A fault diagnosis approach using SVM with data dimension reduction by PCA and LDA method, Chinese Automation Congress (CAC), pp. 869 – 874, (2015).

AUTHORS



Mohamed Denguir received his bachelor and master degree in Electrical engineering, Electronics and Information technology (EEI) from the University of Erlangen-Nuremberg (FAU) in 2013 and 2015. His specialization were microelectronic in his bachelor and automation technology (control engineering) in the master degree. He is a scientific assistant at the Institute for Reliable Circuits and Systems, University of Erlangen-Nuremberg since 2015. He has published five papers in the fields of: verification and testing of circuits and systems, structure-preserving modelling of circuits, fault diagnosis, test compaction and multidimensional analysis and separation of high-dimensional test results

Sebastian Sattler received the Dipl.-Ing. and Dr.-Ing., both in Electrical engineering and Computer Science, from the Munich University of Technology in Germany, in 1989 and 1994, respectively. From 1996 to 2009, he was at Infineon Semiconductor AG (former Siemens Semiconductor AG) as CAD/CAT Engineer, Manager for Analog Design For Test, and Manager for Acquisition of Public Funding in Testing. Since 2009 he is head of the Institute for Reliable Circuits and Systems at the University of Erlangen-Nuremberg, Germany. He has been engaged in research and development on Analog & Mixed- Signal Design for Test applications and techniques, integrated into SOC/SIP for communication and automotive systems. In these fields, he has published about 60 papers and formed more than 30 patents. In 2005, he received the German EDA Achievement Award. His fields of activities include verification, production testing and life time diagnosis for analog, mixed-signal and digital circuits.