# New Methods and Coverage Metrics for Functional Verification

Vasco Jerinić Jan Langer* Ulrich Heinkel* Dietmar Müller*
CONSOLEDA GmbH, Chemnitz
*Circuit and System Design Group, Chemnitz University of Technology
Germany
email: vje@infotech.tu-chemnitz.de

## Abstract

*An ever increasing portion of design effort is spent on functional verification. The verification space as the set of possible combinations of a design's attributes is likely to be very large making it infeasible to verify each point in this space. State-of-the-art verification tools tackle this problem by using directed random generation of combinations in conjunction with manually defined corner cases in order to get satisfactory coverage with the desired distribution. In this work, the underlying methodology to automatically generating complete sets of disjoint coverage models on the basis of formal attribute definitions is extended to take relational constraints into account. This allows the utilization of coverage models with non-orthogonal, non-planar boundaries, which can make hole analysis for coverage data obsolete. It shall be demonstrated, how the proposed methodology can be used to automatically determine corner cases more accurately than it is possible with conventional approaches.*

## 1. Introduction

Functional verification takes an increasing portion of the effort in hardware design. The verification space of a reasonably large design as the set of all possible combinations of a design's attributes, i.e. a design's parameters and inputs [16], is likely to be very large which makes it infeasible to verify all combinations. Verification environments, e.g. *Specman Elite™* from *Cadence* [17] or *VERA™* from *Synopsys* [18], use randomly generated inputs in conjunction with the special verification of manually defined corner cases to achieve an acceptable coverage of the verification space with the desired distribution.

Typically, a special hardware verification language (HVL), e.g. *e* or *OpenVERA*, is used to implement verification by driving stimulus data into the hardware description language (HDL) model or netlist and collecting its response. Apart from comparing the output to expected data, coverage data for post simulation processing can be collected. Functional coverage focuses on a design's functionality, as opposed to code coverage which tries to ensure that every line of source code is executed at least once. Therefore, verification strategies using functional coverage analysis in order to redirect the verification process are design and even implementation specific [16, 21].

## 2. Previous Work

When using functional coverage, coverage models containing a large set of verification (coverage) tasks are systematically defined by the user and then used to measure, monitor and redirect the testing process [16]. Each model covers a specific area of the verification space [13] by restricting it according to defined boundaries [6]. In order to cover the whole verification space, every point of the space must be handled by one of the coverage models. Every combination of attribute values, i.e. each element of the cross-product of a model's attributes, may be either legal or illegal, which makes restrictions describing valid combinations indispensable. This part of the coverage models is most difficult to define, since often a deep understanding of the specification, design and implementation is needed to create correct restrictions [16]. The PARAGRAPH methodology [11] uses so-called *Domain Graphs (DG)* to split the verification space into subspaces called domains on the basis of formal descriptions of attributes and their interdependencies. DG are constructed by removing all invalid, i.e. forbidden attribute combinations and unifying combinations which are virtually equal [9]. An example of combinations leading to identical behaviour are different parity bit positions in a serial bit stream while parity generation is turned off. From DG disjoint coverage models are automatically derived.

This work extends PARAGRAPH to account for relational constraints and therefore to derive coverage models with non-orthogonal, non-planar boundaries. By this means, a complete set of coverage models to fill the verification space

3-9810801-0-6/DATE06 © 2006 EDAA

is defined, avoiding large cohesive holes in the verification space. Thus, using these models can potentially make hole analysis techniques for coverage data as proposed in [16] obsolete. The reduced verification space must be efficiently represented in order to allow rapid traversion for automatic stimulus generation during simulation. Therefore, the proposed DG extensions are not implemented directly, but mapped to binary diagrams. The corresponding data structures were derived from *Binary Decision Diagrams* and underwent an evolution over *Multi-terminal Binary Decision Diagrams* and *Binary Moment Diagrams* to *Kronecker Multiplicative Binary Moment Diagrams* [1, 2, 3, 4, 5, 20]. The latter allow the representation of most expressions and their interdependencies in linear size.

## 3. Verification Space

A design's simulation stimulus consists of a set of stimuli vectors $M = \{a_1, a_2, \ldots, a_m\}$, where each attribute $a_i$ can take any value inside its defined range $A_i$. The verification space can then be constructed as $V = A_1 \times A_2 \times \ldots \times A_m$. Each element $v = \langle a_1, a_2, \ldots, a_m \rangle$ of $V$, i.e. each point in the verification space, represents one combination of attribute assignments. It can be either legal or illegal, depending on additional attribute interdependencies. A combination $v$ is valid in case all defined interdependencies are fulfilled. The set of valid combinations shall be denoted as $G \subseteq V$.

The representation of $G$ as DG was limited to orthogonal subspaces, i.e. only subspaces with planar boundaries parallel to the attribute axes could be expressed [8]. The extension presented in this work allows the representation of subspaces with non-orthogonal and even non-planar boundaries. Suppose a set of attributes $M = \{a, b\}$, with the ranges $A = \{1 \ldots 10\}$ and $B = \{1 \ldots 10\}$. Two additional interdependencies are expressed as relational constraints

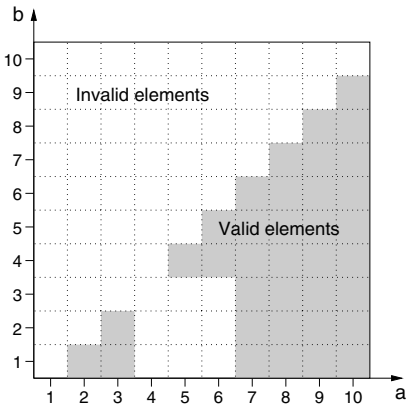


**Figure 1. 2-dimensional verification space**

$$a > b \tag{1}$$

and

$$(a-5)^2 + (b-2)^2 > 2. \tag{2}$$

Fig. 1 shows $V$ and the effective verification space $G \subseteq V$ due to the interdependencies. The set $G$ can be represented as *Multi-valued Decision Diagram (MDD)* [12] with the characteristic function

$$f(a_1, a_2, \ldots, a_m) = \begin{cases} 1: & \langle a_1, a_2, \ldots, a_m \rangle \in G \\ 0: & else \end{cases}.$$

Fig. 2 shows the corresponding MDD. The internal representation relies on special binary decision diagrams (K*BMD) which allow to fulfill both – usually contradictory – demands, compact storage and efficient manipulation [20]. Each path from the root to a leaf node in the MDD represents a subspace of $G$. A particular attribute combination $v = \langle a_1, a_2, \ldots, a_m \rangle$ can be selected by assigning values to the attributes according to the edge values in the MDD.

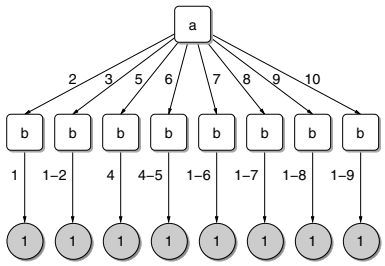


**Figure 2. Multi-valued Decision Diagram**

Given a fixed set of MDD operations – one for each arithmetic or logic operator available for the formal attribute description –, two basic graph types, identity and constant, are sufficient to construct any MDD. The MDD of a constant consists of a single leaf node carrying the constant value as depicted in Fig. 3 for the constant 5. Identity graphs consist

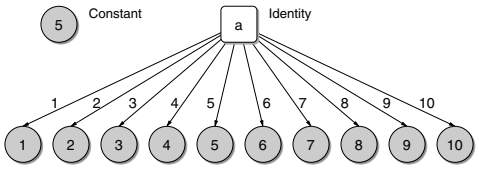


**Figure 3. Constant and identity graph**

of only one attribute node. Every adjacent edge is connected to a leaf node for each value in the attribute's range. The identity graph for attribute $a$ with $A = \{1 \ldots 10\}$ is shown in Fig. 3. The set of operations covers logic and arithmetic operations supported by a number of HDL and HVL, such as *VHDL, Verilog* and *e* [7]. The MDD of $G$ is constructed by iteratively applying MDD operations to the basic graphs. First, a MDD is constructed for each relational constraint. Since all constraints must always be fulfilled, the distinct

MDD must be linked via logical conjunction in a second step in order to get the final MDD representing $G$. Fig. 4 shows the different steps executed during the MDD construction of the second relation (cf. Eq. (2)) of the example above.

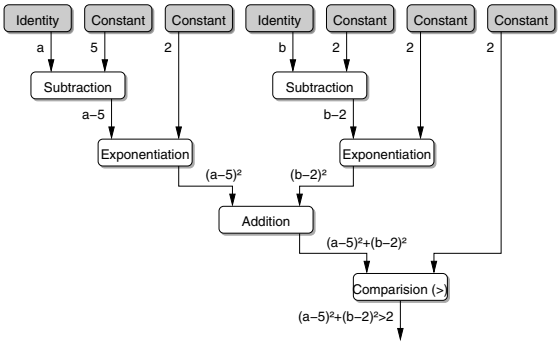


**Figure 4. Iterative construction of MDD**

## 4. Cardinality

It shall be shown in the following section, that it is essential to determine the absolute number of elements in the effective verification space. This value is equivalent to the cardinality $|G|$ of the set $G$. The cardinality of MDD can be computed by counting the number of possible paths from each node and recursively accumulating the values of all adjacent nodes. Fig. 5 shows the value for each distinct node and the accumulation result of 36, which can be validated by counting the grey boxes printed in Fig. 1.

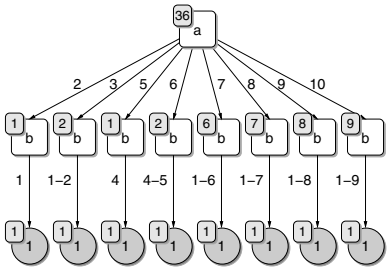


**Figure 5. MDD cardinality**

## 5. Traversion

As stated in section 3, valid attribute combinations can be selected by choosing one path in the MDD of $G$ and assigning attributes according to the edge values. Hence, simulation stimuli can be generated by randomly traversing the MDD. Starting at the root node, one of the adjacent edges is selected at random and an arbitrary value inside the edge's range is assigned to the corresponding attribute. This procedure is repeated until a leaf node is reached. Note that the assigned values do not require any further checking, since the MDD contains valid attribute combinations only.

An equal distribution of the attribute combinations over the verification space is (for most purposes) indispensable. Therefore, it must be ensured that each combination is selected with the same probability. This is accomplished by using the ratio of the target and source node cardinalities as edge weights and selecting edges according to their weight. Fig. 6 shows the MDD of the example with all nodes having their cardinalities displayed in order to explain the usage of edge weights in detail. Consider attribute combination

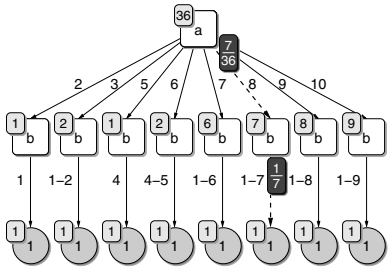


**Figure 6. MDD traversion**

$\langle a,b\rangle = \langle 8,3\rangle$ . Since every combination shall be selected with the same probability, the probability of selecting this one out of an overall number of 36 must be $\frac{1}{36}$. The source and target node cardinalities of edge $\overset{8}{\longmapsto}$ leaving node $a$ are 36 and 7 respectively. Thus, this edge assigned a weight of $\frac{7}{36}$. Value 3 from the adjacent edge $\overset{1-7}{\longmapsto}$ of the former target node $b$ is selected with a probability of $\frac{1}{7}$ according to the corresponding source and target node cardinalities. By this means, the probability of selecting combination $\langle 8,3\rangle$ computes to $\frac{7}{36}\cdot\frac{1}{7}=\frac{1}{36}$ as postulated.

Erroneous behaviour due to specification misinterpretation or design bugs often occur in conjunction with input vectors taking marginal values. Therefore, the opportunity to direct random generation to emphasize those corner cases is desirable, i.e. to select points close to the boundaries of the effective verification space $G$. Using one of the verification environments commercially available, users can manually define subranges to be emphasized for every attribute. In case *Specman Elite*$^{TM}$ is used, the following *e* statements could be used to emphasize the maximum and minimum value of the example attributes $a$ and $b$ by a factor of 5 :

```
keep soft me.a == select {
 5: [1];   // min value of attribute a
 5: [10];  // max value of attribute a
 1: others;};
keep soft me.b == select {
 5: [1];   // min value of attribute b
 5: [10];   // max value of attribute b
 1: others;};
```

While these definitions can adequately be applied to the verification space $V$, they lead to missing corner cases if used on the restricted effective verification space $G$ as depicted in Fig. 7. The points of $V$ covered by the manual corner definitions ($a = \{1,10\}$ , $b = \{1,10\}$) are marked with X, the space $G \subseteq V$ is printed in light grey. Since random gen-

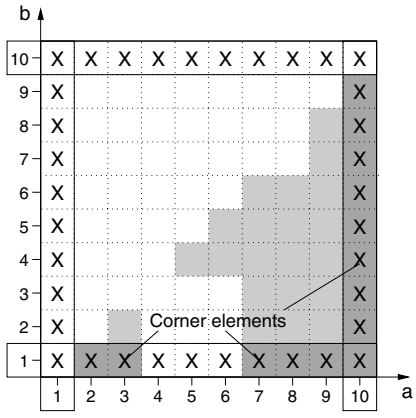


**Figure 7. Inadvertently missed corners**

eration must take the relational constraints into account as well, only the intersection of both sets is emphasized. It can be seen that some points on the boundary of $G$ will – in contradiction to the user's intent – not get emphasized at all. By deriving corner case definitions from $G$, this deficiency can be remedied. Each point of $v \in G$, that has at least one neighbour $v' \notin G$ , is a point on the boundary of $G$. Two points $v = \langle a_1, a_2, \ldots, a_m \rangle$ and $v' = \langle a'_1, a'_2, \ldots, a'_m \rangle$ are regarded as neighbours in case their *manhattan distance*, i.e. the sum of the single attribute differences, does not exceed the value of 1 :

$$\begin{array}{l} \textit{Manhattan distance:} \\ d(v,v') = \sum_{i=1}^{m} |a_i - a'_i|. \end{array}$$

In case not only points on the boundary itself, but points close to it are to be selected, the stimulus generator can generate appropriate attribute combinations by selecting points with a manhattan distance to the boundary smaller than the desired corner width $W$. Fig. 8 shows the correct set of corner points for a corner width $W = 1$.

## 6. Functional Coverage

Modern verification strategies collect functional coverage data to monitor the verification progress and to redirect the verification process in order to improve functional coverage. The coverage grade is computed as the fraction of valid attribute combinations already generated during simulation. To perform this computation, the number of distinct combinations generated so far, and the overall number of

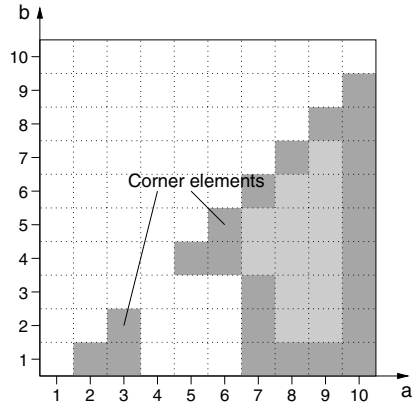


**Figure 8. Complete set of corners ($W = 1$)**

possible (valid) combinations is required. The former can be determined by looking at coverage data collected during simulation, which can as well be used to unveil large cohesive areas in the verification space not covered by simulation yet. The latter, i.e. the cardinality $|G|$ , is not available to state-of-the-art verification flows, since they usually check the validity of combinations 'on-the-fly' during stimulus generation instead of constructing some representation of $G$ in advance. On that score, these tools use the cardinality $|V|$ instead of the – in case of additional constraints usually much smaller – $|G|$ , which leads to an underestimation of coverage grades.

The set of combinations covered by simulation, i.e. the coverage space, shall be denoted as $C \subseteq G$ and is stored as additional MDD. After generating an attribute combination $v = \langle a_1, a_2, \ldots, a_m \rangle$, it is added to $C$ :

$$C \leftarrow C \cup v.$$

To illustrate this process, 10 combinations were randomly generated and accumulated in $C$ : $\langle 6,5 \rangle$, $\langle 9,5 \rangle$, $\langle 8,5 \rangle$, $\langle 10,4 \rangle$, $\langle 10,1 \rangle$, $\langle 3,2 \rangle$, $\langle 9,3 \rangle$, $\langle 8,5 \rangle$, $\langle 8,6 \rangle$ and $\langle 8,4 \rangle$. Fig. 9 shows the corresponding MDD. The coverage grade in this

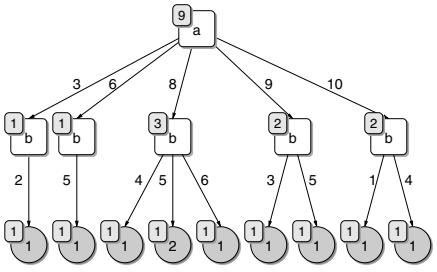


**Figure 9. MDD of the coverage space $C$**

case calculates to

$$\frac{|C|}{|G|} = \frac{9}{36} = 0.25,$$

as opposed to the computation using $|V|$ instead of $|G|$, which yields

$$\frac{|C|}{|V|} = \frac{9}{100} = 0.09.$$

The MDD representation of $C$ facilitates computation of cross coverage grades in a post simulation process. No declaration of cross coverage items in advance nor information about the time at which attributes took a certain value is necessary. The only user action required is to define the set of attributes $U \subseteq M$ to include into cross coverage computation. In a so-called *smoothing* process, every attribute not contained in $U$ is removed from $C$ and $G$. The cross coverage can then be computed on the basis of the resulting sets $C_U$ and $G_U$. In case $U$ consists of only one attribute $a_j$, the coverage grade of that single attribute is computed instead. Fig. 10 opposes spaces $G$ and $C$ to depict the smoothing process and the resulting coverage grades.

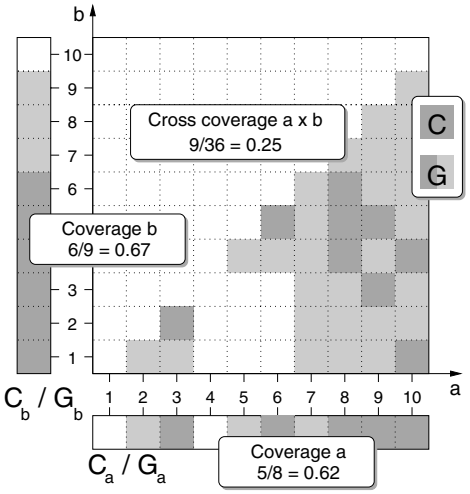


**Figure 10. Coverage and cross coverage**

In order to determine the coverage of the example attribute $b$, smoothing with respect to $a$ is necessary, which means row-wise subsumption, while smoothing with respect to $b$ means column-wise subsumption. Having a representation of $G$ available, the accumulation of generated combinations as coverage space $C$ is sufficient to determine any coverage or cross coverage possibly desired, without the necessity to declare any coverage items in advance.

## 7. Results

The software package PARAGRAPH [10] was extended with the described capabilities. It reads formal attribute definitions from a file using a special language resp. file format proposed in [11] and constructs the corresponding MDD. A bi-directional serial communication interface developed with an industrial partner during an earlier project shall be used as demonstration example. 30 attribute definitions (Table 1) were extracted from the HDL model. Using the basic

| attribute | cardinality | attribute | cardinality |
|---|---|---|---|
| PINMODE | 2 | RXWA1_LEN | 2 |
| TXPCM | 2 | RXINTADDR | 32 |
| RXPCM | 2 | TXEDGE | 2 |
| TXCLKSEL | 4 | TXDEL | 2 |
| RXCLKSEL | 4 | TXPOL | 2 |
| CLK0SEL | 3 | TXPERIOD | 3 |
| CLK1SEL | 3 | TXWIDTH | 5 |
| RXEDGE | 2 | TXALIGN | 2 |
| RXDEL | 2 | TXMONO | 3 |
| RXPOL | 2 | TXMUTE_L | 2 |
| RXPERIOD | 3 | TXMUTE_R | 2 |
| RXWIDTH | 5 | TXCLK0_OUT | 2 |
| RXALIGN | 2 | TXCLK0_CONT | 2 |
| RXCLK1_OUT | 2 | TXWA0_LEN | 2 |
| RXCLK1_CONT | 2 | TXINTADDR | 32 |

**Table 1. Set of attributes**

DG methodology of the preceding PARAGRAPH version, the verification space can be split into 64 domains while being limited to `REQUIRE` and `CONFLICT` statements. The fact that attribute `RXEDGE` only influences the interface behaviour in case `RXPCM` is assigned the value `off` is for instance declared using a `REQUIRE` statement:

```
REQUIRE RXEDGE:
    RXPCM name:off;
```

This way another 29 attribute interdependencies could be defined reducing the overall number of valid attribute combinations by a factor of 2846. Using the extended MDD methodology proposed here, 6 additional relational constraints, such as

```
RELATION TXPERIOD >= 2*TXWIDTH;
```

can be declared. The generation of the corresponding MDD on a *Sun UltraSPARC®10* takes about 73 seconds. The effective verification space $G$ is reduced by a factor of 17254 in comparison to the complete verification space $V$. Table 2 lists a number of designs to which the proposed methodology has been applied. Beside the equivalent gate count of the particular design, the number of lines, attributes, and constraints in the formal attribute description, the number of nodes in the resulting DG, the time needed for the construction of the DG including the MDD representing the relational constraints, and the resulting reduction in size of the effective verification space are given.

| design | source | gates | lines | attributes | constraints | DG nodes | construction | reduction |
|---|---|---|---|---|---|---|---|---|
| bcdbin | opencores [19] | 1500 | 49 | 3 | 16 | 32 | $2s$ | 77 |
| fifo | opencores [19] | 30000 | 21 | 3 | 6 | 34 | $3s$ | 4 |
| vga | internal | 3000 | 52 | 9 | 15 | 146 | $7s$ | 32 |
| moments | internal | 70000 | 41 | 9 | 12 | 223 | $17s$ | 1413 |
| fsi | industrial | 15000 | 84 | 15 | 16 | 60 | $3s$ | 239 |
| i2s | industrial | 45000 | 194 | 30 | 36 | 967 | $73s$ | 17254 |

**Table 2. Construction time and reduction factor**

## 8. Conclusion

This work enhances the basic PARAGRAPH methodology to automatically derive complete sets of functional coverage models from formal attribute descriptions. With the extensions proposed, PARAGRAPH is enabled to utilize MDD to take relational attribute constraints into account in order to split the verification space into disjoint coverage models with non-orthogonal, non-planar boundaries. By this means, the technique to prevent the traditional coverage grade underestimation of conventional approaches is refined by relying on an analytical model of the verification space. The automatic extraction of corner cases with respect to the model boundaries is raised to a whole new level, since the completeness of the set of boundary points can be guaranteed for the first time.

Current work focuses on the representation of transitions between different functional states or stimuli sequences. Additional research is necessary to investigate how these aspects can be expressed in terms of decision diagrams or graphs, e.g. *Temporal Event Relation Graphs (TERG)* [15] or *Hierarchical Temporal Event Relation Graphs (HiTER)* respectively [14]. On that basis, solutions to automatically drive a design into a certain state and efficiently direct verification from that state could be developed.

## References


[1] S. B. Akers. Binary Decision Diagrams. *IEEE Transactions on Computers*, C-27(6), June 1978.

[2] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic Decision Diagrams and Their Applications. In *International Conference on Computer Aided Design (ICCAD '93), Santa Clara, California*, pages 188–191. ACM/IEEE, IEEE Computer Society Press, Nov. 1993.

[3] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, Aug. 1986.

[4] R. E. Bryant and Y.-A. Chen. Verification of Arithmetic Functions with Binary Moment Diagrams. Technical Report CMU-CS-94-160, Carnegie Mellon University, 1994.

[5] E. M. Clarke, M. Fujita, and X. Zhao. Hybrid decision diagrams overcoming the limitations of MTBDDs and BMDs. In *International Conference on Computer Aided Design (ICCAD '95), Los Alamitos, CA, USA*, pages 159–163, 1995.

[6] D.P. Appenzeller and A. Kuehlmann. Formal Verification of a PowerPC Microprocessor. *IEEE International Conference on Computer Design*, ICCD('95), 1995.

[7] V. Jerinić. *ParaGraph - Parameter checking for Intellectual Properties*. PhD thesis, Chemnitz University of Technology, 2005. (Published in German).

[8] V. Jerinić and D. Müller. Shrinking the Parameter Space of IP utilizing Parameter Domains. *International Workshop on IP Based SoC Design*, Grenoble(France):109–113, 2002.

[9] V. Jerinić and D. Müller. Assertion-Based Parameter Checking for IP. *System-on-Chip and ASIC Design Conference 2003*, Santa Clara(CA, USA), Jan. 2003.

[10] V. Jerinić and D. Müller. Tool-Demo: ParaGraph - Parameter checking for IP. *Presentation of the edacentrum Design, Automation and Test in Europe Conference (DATE)*, page , Mar. 2003. Munich.

[11] V. Jerinić and D. Müller. Safe integration of parameterized IP. *INTEGRATION, the VLSI journal*, 37:193–221, 2004.

[12] T. Y. K. Kam. Multi-valued decision diagrams. Master's thesis, University of California, Department of Electrical Engineering and Computer Sciences, Berkeley, CA 94720, USA, 1990.

[13] K. Kohno and N. Matsumoto. A New Verification Methodology for Complex Pipeline Behavior. *Design Automation Conference*, DAC:816–821, 2001.

[14] Y.-S. Kwon, Y.-I. Kim, and C.-M. Kyung. Systematic functional coverage metric synthesis from hierarchical temporal event relation graph. In *Design Automation Conference*, pages 45–48, New York, NY, USA, 2004. ACM Press.

[15] Y.-S. Kwon and C.-M. Kyung. Functional Coverage Metric Generation from Temporal Event Relation Graph. In *Design, Automation and Test in Europe (DATE) 2004*, pages 670–671. IEEE Computer Society, IEEE, Feb. 2004.

[16] O. Lachish, E. Marcus, S. Ur, and A. Ziv. Hole Analysis for Functional Coverage Data. *Design Automation Conference*, DAC:807–812, June 2002.

[17] Cadence Design Systems Inc. *http://www.cadence.com*.

[18] Sysnopsys Inc. *http://www.synopsys.com*.

[19] Opencores. *http://www.opencores.org*.

[20] R. Drechsler, B. Becker, and S. Ruppertz. K*BMDs: a new data structure for verification. In *IFI WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods, Frankfurt, Germany*, 1995.

[21] S. Ur and A. Ziv. Off-The-Shelf Vs. Custom Made Covergae Models, Which Is The One for You? In *Software Testing, Analysis, and Review (STAR98)*, May 1998.