Using Satisfiability in Application-Dependent Testing of FPGA Interconnects

Mehdi Baradaran Tahoori
Center for Reliable Computing
Stanford University
mtahoori@crc.stanford.edu

ABSTRACT

In this paper, a new technique for testing the interconnects of an arbitrary design mapped into an FPGA is presented. In this technique, only the configuration of logic blocks used in the design is changed. The test vector and configuration generation problem is systematically converted to a satisfiability (SAT) problem, and state of the art SAT-solvers are exploited for test configuration generation. Experimental results on various benchmark circuits show that only two test configurations are required to test for all bridging faults, achieving 100% fault coverage, with respect to the fault list.

Categories and Subject Descriptors
B.8.1 [**Reliability, Testing, and Fault-Tolerance**]

General Terms
Reliability, Verification.

Keywords
Field-Programmable Gate Array, Interconnect.

1. INTRODUCTION

SRAM-based field programmable gate arrays (FPGAs) are two-dimensional arrays of logic blocks and programmable switch matrices, surrounded by programmable input/output blocks on the periphery. FPGAs are widely used in many applications such as networking and adaptive computing, due to their reprogrammability, reduced design cycle and time-to-market compared to conventional ASIC design flow.

More than 80% of the transistors in an FPGA are used in the interconnect network. Also, more than eight metal layers are used for the wiring channels in the interconnect network. Hence, FPGAs are vulnerable to bridging faults in the interconnects. Test generation for bridging faults is more complicated than traditional testing for stuck-at faults. Note that stuck-at model in the interconnects can be modeled as bridging to power rails.

Testability issues are not considered in the design flow using FPGAs, as FPGA users typically rely on manufacturing test of FPGAs. There are no scan chains, BIST circuitry, or test points in typical FPGA-based designs. Hence, these designs are not fully testable using conventional ASIC ATPG tools.

On the other hand, some FPGA chips that do not pass manufacturing test may still be usable for some specific designs. In this case, the defects are located in some areas of the chip which are not used by a particular design. By testing the resources of an FPGA with respect to a specific design to be implemented on it, some faulty chips can be sold to customers. These FPGAs, which are good only for particular designs and do not have general programmability of typical FPGAs, are called *application-specific FPGAs* (ASFPGAs). ASFPGAs are profitable for relatively large volume designs which have been completely finalized, i.e. the final placed and routed version is fixed.

Based on these mentioned facts, FPGA vendors can benefit from application-dependent testing of FPGAs, which tests only the FPGA resources used on a particular design, in order to increase the yield and make money from those chips that are previously marked as rejected parts. This strategy has been adopted by Xilinx [Xilinx 02].

Application-dependent testing of FPGAs is described in [Das 99]. In this technique, every CLB used in the mapped design is reconfigured as transparent logic followed by flip-flips in order to construct scan chains. Also, fanout branches of a net are tested in different test configurations, i.e. dependent logic cones are tested in different configuration, resulting in a few number of test configurations. Due to complexity of configuration generation algorithm, it cannot be applied to large designs.

In this paper, a new testing technique for application-dependent testing of bridging faults in FPGA interconnects is presented. We consider both wired-OR and wire-AND bridging fault models [Chess 98] [Storey 98]. In the presented method, only the logic blocks of the FPGA used by the mapped design are reprogrammed to implement some special type of logic functions. Hence, no extra placement and routing are necessary for test configuration generation. The test configuration generation problem is systematically converted into a satisfiability (SAT) problem, and state-of-the-art SAT-solvers are exploited to solve this problem.

The logic implemented in the logic blocks can be tested using well-known techniques for testing logic blocks [Stroud 96][Renovell 00]. Hence, it is not covered in this paper.

The rest of this paper is organized as follows. In Sec.2, the notion of single-term functions and testability of logic networks are presented. In Sec. 3, testing techniques for bridging faults are presented. In Sec. 4, the systematic approach to convert the configuration generation problem into a SAT problem, along with generalization to arbitrary fault list, is presented. In Sec. 5, the experimental results are presented. Finally Sec. 6 concludes the paper.

2. TESTABILITY OF LOGIC FUNCTIONS

In this section, some specific logic functions are introduced to be implemented in the logic blocks of mapped designs.

Consider a logic function with *m* inputs. Each of the 2^m input combinations corresponds to a *term* in the truth table of the function. A *single-term* function is a logic function which has only one minterm or only one maxterm. In other words, the truth table for a single-term function, *F*, has only one row with $F = 1$, or only one row with $F = 0$. The input corresponding to this specific minterm or maxterm is called the *activating input*. An example of a single-term function is shown in Fig. 7.1, where the activating input is applied. This function has only one minterm, A′BC′D′. Note that A/1 (A stuck-at-1 fault) is detectable under the applied input pattern, as the faulty output is 0**,** which is unequal to fault-free output, 1. Similarly, B/0, C/1, and D/1 are also detectable. Consider the bridging fault between A and B (denoted by $A_{BF}B$). In the case of wired-and (WAND) model, both inputs become 0, and the effective input is $ABCD = 0000$, and faulty output is $F(0000) = 0$. Otherwise, in the case of wired-or (WOR), both inputs become 1 and faulty output is $F(1100) = 0$. In both cases, the bridging fault is detectable. Similarly, $B_{BF}C$ and $B_{BF}D$ are also detectable.

A fault on a node is *sensitized* if the applied test vector sets that node a logic value opposite to the fault value. The following theorem generalizes the above example and explains the conditions for detectability of faults in single-term functions.

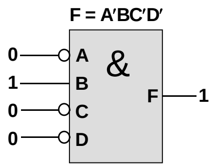

Figure 1 A single-term function with activating input pattern

Theorem 1. For a single-term function *F*, if the applied input vector is the activating input *V*, all sensitized stuck-at and bridging faults are detectable.

Proof. Consider a fault *f* (stuck-at or bridging fault) such that it is sensitized, i.e. if *f* is *n/v*, *n* is set to *v′* by applying *V*, and if *f* is $a_{BF}b$, *a* and *b* have different values in *V*. In order to detect *f*, it is only sufficient to propagate the fault to the output. Since the fault is already sensitized, the fault term, the term corresponding to the faulty inputs, C_f, is different from the original term, C_V. Because the original term is the activating term, the value of the fault term must be different from the value of original term, $C_V \neq C_f$, as *F* is single-term. Hence, the fault-free output is different from the faulty output and the fault is detectable.

Theorem 2. Consider a network of single-term functions *N*, and the input pattern *V*, such that the inputs of every block is the activating input of that block. Then, all the activated faults are detectable. In other word, for every net *n* with value V_n, *n* stuck-at V_n' is detectable, and for each pair of nets, n_i and n_j, with $V_{ni} \neq V_{nj}$, the bridging fault between n_i and n_j is detectable.

Proof. In this general case, the propagation path from the fault site to the primary output (PO) may consist more than one block. Consider a propagation path from the fault site to a PO. Based on Theorem 1, the fault effect appears on the output of the first block in the path, as a fault in the input of the second block in the path. Based on an induction on the length of the path, the fault reaches to the PO. Hence, the fault can be detected.

Figure 2 shows an example of a network of single-term functions. All the activated faults are detectable.

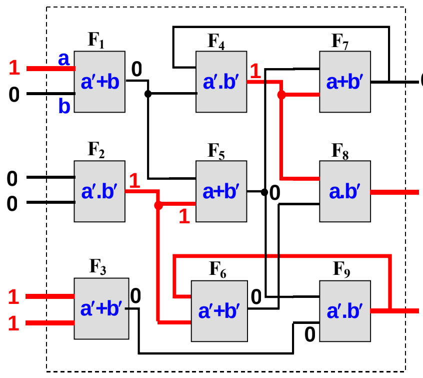

Figure 2 A logic network of single-term functions

Note that if the network is sequential, i.e. some single-term functions are followed by flip-flops, the above theorem is still applicable. In this case, the preset value of the flip-flop must be equal the value of the net connected to its D input, as defined in the conditions of Theorem 2. For example, if F_5 is followed by a flip-flop in Fig. 2, the preset value of that flip-flop must be 0, the value of the output of F_5. In this case, the required number of test clock cycles is equal to the maximum *sequential depth* of the network, the maximum number of flip-flops on a path from a PI to a PO. The test vector must remain unchanged during all these test clock cycles.

Theorem2 gives a sufficient condition for detection of all activated faults in a logic network of single-term functions. The following theorem expresses the necessary condition for this detectability.

Theorem 3. Consider an arbitrary network of (combinational) logic functions, *N*, and a given test vector ***V***. If for every net *n* with value V_n, *n* stuck-at V_n' is detectable, then all the functions in the network must be single-term functions whose inputs are the activating inputs.

Proof. Consider an arbitrary function *F* in the network *N*. Let say the inputs of *F* under the primary input vector ***V*** is *v*. In the presence of any multiple (including single) stuck-at faults in the input pins of *F*, the actual input values seen by *F* change from *v* to a faulty value *v′*. Since all the possible stuck-at faults must be detectable, all combinations of input values other than *v* can occur as faulty inputs of *F*. So, if *F* has *m* inputs, *v′* can take all $2^m - 1$ possible combinations other than *v*. The necessary condition to detect each fault in the inputs of *F* is to propagate it to the output of *F*. Hence $F(v') \neq F(v)$ for all $2^m - 1$ possible values for *v′*. This implies that *F* is single-term and *v* is the activating input. Since we chose *F* as an arbitrary function in *N*, this property must hold for all the functions in *N*, so the proof is complete.

3. TESTING FOR BRIDGING FAULTS

3.1 Fault List

It is not appropriate to consider bridging faults between all possible pairs of nets in the design. First, the size of the fault list becomes intractable. Second, bridging faults between some pairs of nets are improbable due to physical neighborhood obtained from layout information. Inductive fault analysis (IFA) techniques are proposed to extract the fault list from the physical layout information [Ferguson 88]. These techniques are very time consuming for a medium to large size designs.

The presented technique supports both internally generated fault list for bridging fault, or any user-specified fault lists. All the nets in the design are partition into some groups. All the nets in the same group are tested for all possible pair-wise bridging faults. A group of *m* nets has *m(m-1)/2* bridging faults. The bridging fault between two nets in different groups is not included in the fault list.

In the internally generated fault list, all the inputs of a look-up table (LUT) are considered in the same group. Hence, the number of the groups is equal to the number of LUTs used for the design. As a result, the size of the fault list is a constant factor of the number of logic blocks used in the design. Note that this fault list exploits physical neighborhood, since the nets in different blocks have less probability of bridging faults than those in the same block.

3.2 Test Configurations

For the FPGA with *n*-input LUTs, log_2n test configurations with single-term functions are sufficient to cover all bridging faults with respect to the fault list described in Sec. 3.1. The activating inputs are columns of binary numbers using log_2n bits for n-input LUTs (so-called Walsh codes).

Consider the binary representation of m^{th} input of the LUT using log_2n bits. Each bit position corresponds to one of the log_2n test configurations, as follows. If the i^{th} bit position is 1 for the m^{th} input, it means that the m^{th} bit of the activating input is 1 in the i^{th} configuration, otherwise 0.

The configuration of the LUT is determined by its activating input and the value of the output for that input (single-term function). Figure 3 shows an example for a 4-input LUT, with the activating inputs and single-term functions. Also, the bridging faults detected by each configuration are shown. For example, input C is coded by 10, using two bits ($log_2 4$). The corresponding bit of the activating input is 1 in the first configuration and 0 in the second configuration.

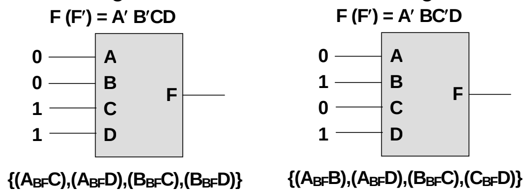
F (F′) = A′ B′CD
0
0
1
1
A
B
C
D
F
{(A_BF C),(A_BF D),(B_BF C),(B_BF D)}
F (F′) = A′ BC′D
0
1
0
1
A
B
C
D
F
{(A_BF B),(A_BF D),(B_BF C),(C_BF D)}

Figure 3 Two configurations for a 4-input LUT

Theorem 4. The above technique detects all bridging faults among all pairs of nets in the inputs of an n-input LUT, in $log_2 n$ configurations.

Proof. Consider two arbitrary inputs of the LUT, i^{th} and j^{th} inputs. Let $i<m>$ be the value of m^{th} bit of i. Since i and j are two different inputs, $i \neq j$, they have different binary representations. This means than there should be at least one bit position p, in which $i<p> \neq j<p>$. This implies that in the p^{th} configuration, the i^{th} and j^{th} bits of the activating input have different values. Hence, the bridging fault between these two inputs is activated in the p^{th} test configuration. Based on Theorem 2, the bridging fault between these two inputs is detected.

4. SATISFIABILITY FORMULATION

4.1 Configuration for The Entire FPGA

The previous section presented the configuration generation technique for only one LUT. In this section, an algorithmic approach is presented to generate the test configurations for the entire FPGA.

Due to fanouts, the desired activating inputs for the LUTs cannot be realized for all LUTs in the design. An example is shown in Fig. 4, in which appropriate activating inputs are assigned for LUT$_1$ and LUT$_2$. Since the stem and fanout branches have the same logic value, the desired activating input, *0011*, cannot be realized at the inputs of LUT$_3$. Therefore, *backtrack* should be performed and the assignments of the inputs of LUT$_1$ and LUT$_2$ must be changed in order to obtain a feasible assignment for inputs of LUT$_3$. The problem of configuration generation for the entire FPGA using the presented approach is NP-Complete in general.

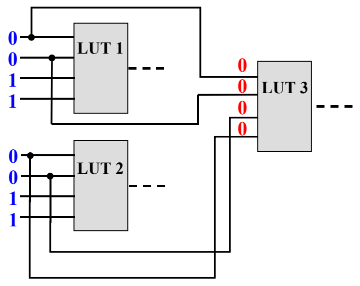

Figure 4 A portion of a design implemented on an FPGA.

In the presented approach, all the nets in the design are partitioned into two groups, *plus* and *minus*, such that for every LUT L with n inputs to be partitioned, $n/2$ of the inputs are assigned into plus group, and the other $n/2$ inputs are assigned to minus group. This process is called *dichotomization*. The fanout branches and stem of the same net must be assigned to the same group.

By assigning 0 (1) to all the nets in the plus group and 1 (0) to all the nets in the minus group, the desired activating inputs for all the LUTs will be obtained. Based on the logic value of the activating input and the output of the LUTs, the configuration of the LUT and the preset value of the flip-flops are determined. The logic values of the nets corresponding to primary inputs are the test vectors.

For the next test configuration, we set $n = n/2$, and the same dichotomizing algorithm is performed for each of the groups obtained in the previous configuration. This process recursively continues for all the $log_2 n$ test configurations.

4.2 SAT Formulation for Grouping Problem

We present a technique to convert the configuration generation problem into a satisfiability problem by constructing a Boolean function $\mathcal{F}$. A solution to $\mathcal{F}$=1 can be converted to the desired test vectors and configurations.

Consider the symmetric logic function $S^n_k(x_1, x_2, ..., x_n)$, where $S^n_k = 1$ if and only if exactly k inputs of S^n_k are 1. For example, $S^3_1 = x_1x_2'x_3' + x_1'x_2x_3' + x_1'x_2'x_3$. S^n_k can be recursively defined as follows

$$S^n_k(x_1,\ldots,x_n) = \begin{cases} \prod_{i=1}^{n} x_i & ,k=n \\ \prod_{i=1}^{n} \overline{x_i} & ,k=0 \\ x_1 . S^{n-1}_{k-1}(x_2,\ldots,x_n) + \overline{x_1} . S^{n-1}_{k}(x_2,\ldots,x_n) & ,0<k<n \end{cases}$$

(Π is the logical AND function):

Consider the set of the nets in the design that must be dichotomized. Let M be the number of LUT inputs that participate in this grouping algorithm. $M = N/2^{i-1}$ at i^{th} configuration for N-input LUTs. We assign one variable per each net. All the fanout stem and branches of the same net share the same variable. For each LUT l, let $x^l_1, x^l_2, \ldots, x^l_M$ be the M inputs of l which have to be dichotomized. Let $F_l = S^M_{M/2}(x^l_1, x^l_2, \ldots, x^l_M)$.

The solution for $F_l = 1$ is an equal-size dichotomization of those M inputs of l, a sub-solution to the problem for only one LUT.

The function that is the SAT formulization for the entire FPGA is:

$$\mathcal{F} = \prod_{for\ each\ LUT\ l} F_l \ .$$

In the variable assignment that satisfies $\mathcal{F}$, every variable whose with logic value 0 is put in the minus group, otherwise is assigned to the plus group. An example is shown in Fig. 5. All the nets are assigned with unique variables. For the first configuration, S^4_2 must be used for all LUTs, since $M = 4$. The SAT function for the first configuration is as follows:

$$\mathcal{F} = S^4_2(x_1,x_2,x_3,x_4) . S^4_2(x_5,x_6,x_7,x_8) . S^4_2(x_7,x_9,x_4,x_{10}) . S^4_2(x_4,x_9,x_{10},x_8)$$

$$\text{where: } S^4_2(x_1,x_2,x_3,x_4) = [(\overline{x_1}+\overline{x_2}+\overline{x_3})(\overline{x_1}+\overline{x_2}+\overline{x_4})(\overline{x_1}+\overline{x_3}+\overline{x_4})(\overline{x_2}+\overline{x_3}+\overline{x_4})$$
$$(x_1+x_2+x_3)(x_1+x_2+x_4)(x_1+x_3+x_4)(x_2+x_3+x_4)]$$

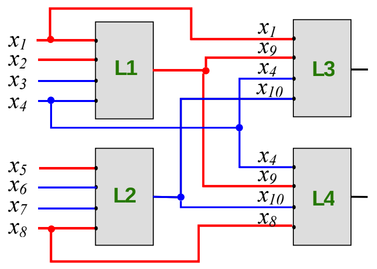

Figure 5 An example design implemented with four LUTs

One variable assignment that satisfies $\mathcal{F}$ is: $<x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8,x_9,x_{10}>$=<0,0,1,1, 0,1,1,0,0,1>. It implies that plus group = $\{x_3,x_4,x_6,x_7,x_{10}\}$ and minus group = $\{x_1,x_2,x_5,x_8,x_9\}$.

For the second configuration, each of the above groups must be partitioned. Hence, there are two SAT functions on two disjoint sets of different variables. These functions are as follows:

$$\mathcal{F}_1 = S_1^2(x_1,x_2).S_1^2(x_5,x_8).S_1^2(x_1,x_9).S_1^2(x_8,x_9)$$

$$\mathcal{F}_2 = S_1^2(x_3,x_4).S_1^2(x_6,x_7).S_1^2(x_4,x_{10}).S_1^2(x_4,x_{10})$$

where $S_1^2(x_1,x_2) = (\overline{x_1}+\overline{x_2})(x_1+x_2)$.

One variable assignment that satisfies $\mathcal{F}_1$ is: $<x_1,x_2,x_5,x_8,x_9>$ = $<1,0,0,1,0>$. Therefore the groups are $\{x_2,x_5,x_9\}$ and $\{x_1\ ,x_8\}$. Similarly, a variable assignment that satisfies $\mathcal{F}_2$ is: $<x_3,x_4,x_6,x_7,x_{10}>$ = $<0,1,0,1,0>$ and the groups are $\{x_3,x_6,x_{10}\}$ and $\{x_4,\ x_7\}$. The results interpreted in terms of the activating inputs and LUT configurations for two test configurations are shown in Fig. 6.

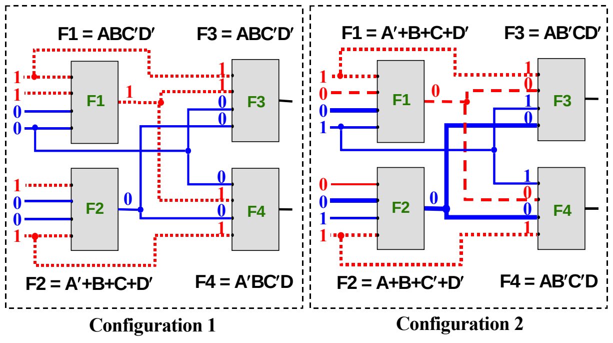

Figure 6 Two configuration for 4-input LUTs

4.3 Generalization to Arbitrary Fault List

The formulation and the technique presented in the previous section can be generalized to any arbitrary user-specified bridging fault list. Typically the nets in the design are partitioned into k groups, $P_1,\ldots,P_k$, such that the bridging fault between two nets, n_i and n_j, is included in the fault list if and only if n_i and n_j belong to the same group.

The SAT formulation (of the first test configuration) for this problem is similar to that presented in Sec. 4.2. Formally, the SAT formulation for each group Pi is:

$$S_{|P_i|/2}^{|P_i|}(x_1,\ldots,x_{|P_i|})$$

where $|P_i|$ is the size of P_i.

Since this condition must be satisfied for all the k groups, the overall SAT function is the logical AND of the above function for all the groups. For the other test configurations, the SAT formulation is similar to the approach presented in Sec. 4.2: recursively partitioning the groups, until the size of each group becomes one in the last configuration. As a result, the total number of required test configurations, N, is given by the following formulae:

$$N = \max_{i=1}^{k}\lceil \log_2 |P_i| \rceil$$

Hence, the total number of test configuration is dominated by the size of the largest fault group.

5. EXPERIMENTAL RESULTS

In order to implement this technique, some FPGA benchmarks are used, which are mapped (placed) designs to a generic switch-based FPGA architecture [Alexander 96]. The SAT-solver used for the implementation is zChaff [Zhang 01]. Table 1 shows the results of test configuration generation method using this SAT-solver for these benchmark designs. The execution time column corresponds to the time spent by the SAT-solver to satisfy the clauses in seconds. As can be seen in this table, for all the benchmark designs, the clauses are satisfiable in less than a second. As a result, the optimum number of test configurations, two, is achievable for all designs. This shows that this technique can be easily used for large designs.

6. SUMMARY

In this paper, a new technique for testing the interconnects of an arbitrary design mapped into an FPGA is presented. Only the configuration of the logic blocks used in the design is changed in order to implement special single-term functions. Formal proofs are provided for 100% bridging fault coverage of the presented technique. SAT formulations are presented for automatic test vector and configuration generation for the entire FPGA. Experimental results on various benchmark designs show that only two test configurations are required to cover all (multiple) bridging faults, achieving 100% fault coverage.

Table 1 SAT formulation results for benchmark circuits

Circuit	CLBs	Nets	Clauses	SAT	Exec.Time	Configs
Term1	90	88	720	√	0.01	2
9symml	110	79	880	√	0.02	2
Appex7	120	115	960	√	0.02	2
busc	156	151	1248	√	0.02	2
Exm2	168	205	1344	√	0.02	2
Alu2	195	153	1560	√	0.02	2
2large	196	186	1568	√	0.02	2
Vda	272	225	2176	√	0.04	2
Dma	288	213	2304	√	0.04	2
Alu4	323	255	2584	√	0.02	2
K2	440	404	3520	√	0.06	2
Bnre	462	352	3696	√	0.04	2
Dfsm	506	420	4048	√	0.04	2
Z03	702	608	5616	√	0.06	2

ACKNOWLEDGMENT

The author would like to thank Professor Edward J. McCluskey for his contributions to this work and supervision of this project.

REFERENCES

[Alexander 96] Alexander, M. J., and Robins, G., *New Performance-Driven FPGA Routing Algorithms*, IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, pp. 1505-1517, 1996.

[Altera 02] *The Programmable Logic Data Book 2002*, Altera Inc., 2002.

[Chess 98] B. Chess, T. Larrabee, *Logic testing of bridging faults in CMOS integrated circuits*, IEEE Trans. Computer, vol.47, pp. 338-345, Mar. 1998.

[Das 99] D. Das, N. A. Touba, *A Low Cost Approach for Detecting, Locating, and Avoiding Interconnect Faults in FPGA-Based Reconfigurable Systems*, Proc. Int'l Conf. On VLSI Design, 1999.

[Ferguson 88] F.J. Ferguson, J.P. Shen, *A CMOS fault extractor for inductive fault analysis*, IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, Vol. 7 Issue: 11 , pp. 1181-1194, Nov. 1988.

[Renovell 00] M. Renovell, Y. Zorian, *Different Experiments in Test Generation for XILINX FPGAs*, Proc. Int'l Test Conf., 2000.

[Storey 98] T. Storey, W. Maly, *CMOS bridging fault detection*, Proc. Int'l Test Conf., pp.842- 851, 1990.

[Stroud 96] C. Stroud, S. Konala, C. Ping, M. Abramovici, *Built-in self-test of logic blocks in FPGAs (Finally, a free lunch: BIST without overhead!)*, Proc. VLSI Test Symp., pp. 387 –392, 1996.

[Xilinx 02] *The Programmable Logic Data Book 2002*, Xilinx Inc., 2002.

[Zhang 01] L. Zhang, C. Madigan, M. Moskewicz, S. Malik, *Efficient Conflict Driven Learning in a Boolean Satisfiability Solver*, Proc. ICCAD, 2001.