MINIMIZATION OF THE HAMMING CODE GENERATOR IN SELF CHECKING CIRCUITS

Pavel Kubalík, Petr Fišer, Hana Kubátová

Department of Computer Science and Engineering
Czech Technical University
Karlovo nam. 13, 121 35 Prague 2
e-mail: xkubalik@fel.cvut.cz, fiserp@fel.cvut.cz, kubatova@fel.cvut.cz

Abstract: The paper focuses on the minimization of the area overhead of check bits generator in the online BIST for circuits implemented in FPGAs. We have used error detection codes (ED codes) to ensure the self-checking property. The newly proposed simplification method consists of converting the duplicate circuit into a two-level network, for which the check-bits are generated. Then the outputs of the circuit are reduced to these check-bits only; the original outputs can be omitted. After that, a multi-level network is synthesized for this circuit. This notion enables us to significantly reduce the resulting logic.

Keywords: FPGA, Built-In Self-Test (BIST), on-line, error detecting codes, self-checking circuit, totally self checking (TSC) circuit

1. INTRODUCTION

Nowadays when the circuit integration increases, the importance of radiation impact on integrated circuits grows even at the sea level. It can affect any circuit used every day. Some machines, like the control units in cars, can play an important role in places such as tunnels, where a car fault can endanger human lives. Other important areas are aviation, medicine or space missions. All of these applications and many others depend on a correct function of circuits and one wrong result can lead to huge losses.

The FPGA circuits are more and more often used to implement any function because of their prices and capabilities to upgrade the function when a bug is discovered. Next advantage is their possible dynamic reconfiguration when a fault in the circuit is detected and localized (XILINX 2003).

This paper is organized as follows. Section 2 describes related works. Section 3 introduces the used fault model and proposes general methodology for comparing the quality of used error detection codes. Section 4 presents the experimental issues and solutions of the parity bits generation for all used benchmark tests. Our future work and conclusions are presented in the Section 5.

2. RELATED WORKS

There are many papers focused on concurrent error detection in a random logic circuit. The combinational circuits are used as basic elements for testing the proposed method ensuring totally self-checking (TSC) properties. There are two basic properties that must be taken into account and which are contradictory: Fault coverage must be achieved as high as possible - up to one hundred percent. The error detecting codes are used to ensure TSC properties. The maximal fault coverage must be ensured for the whole design – for the redundant parts too. All these on-line design methods increase the area overhead of the designed circuit. Area overhead is a second property forcing designers to reach its minimum while saving the maximal fault coverage. There is a relation between the area overhead and the fault coverage. It can be shown that higher fault coverage does not mean higher area overhead, in some cases (Kubalík, 2003).

The concurrent error detection (CED) design methodology used to satisfy TSC property has a deep impact to the fault coverage of circuits implemented in FPGAs. Some results of the fault coverage for circuits implemented in FPGAs are presented in (Bolchini, 2002). Basic methods used for the fault detection in logic circuits are based on the simple duplication. This methodology allows to determine the final area overhead before the duplication generator is executed. The duplicate part can be modified to avoid common-mode failures (CMFs).

Another approach can be used when a duplicate circuit is modified to decrease the number of outputs of a duplicate part (output parity bits are used instead original outputs). The error codes can be used in this case. Both of these techniques are compared in (Mitra, 2000a, b). The area overhead plays an important role in popularity of the method used for ensuring the TSC property. Therefore special schemes were designed for circuits with regular

structures, for example adders, multipliers or memories.

There are two main reasons why the CED techniques were not so popular: a very high area overhead and a low disposition to temporary faults due to their large feature sizes. Nowadays when the deep submicron technology is widely used, incorporating CED techniques into circuits with an unknown structure is more and more important. Some of the new design methods try to reach smaller area overhead but they achieve low fault detection. For example, only some inputs may be used to ensure the partial self-checking property of a multilevel logic, by using low-cost parity error detecting codes (Mohanram, 2003).

Next different design methodology ensuring smaller area overhead uses duplication of only some part of the original circuit. This method is based on reduction of the number of selected input combinations (Drineas, 2003). Some articles describe methods how to detect the faulty part of an FPGA without stopping its function (Abramovici, 1999). These methods test unused part of the FPGA. When the test is performed, the tested part is exchanged with the used part and the testing process is started again for currently unused area. Here the BIST techniques can be successfully exploited (Stroud, 2002).

3. TSC CIRCUIT DESIGN

We have used the structure shown in Fig. 1 as a basic model of the totally self-checking (TSC) circuit. The final scheme consists of four basic blocks: the original combinational circuit, its duplicate, the check bits generator and the checker. The checking bits are generated from the primary inputs of the original circuit. The primary outputs and the checking bits are used as an encoded output. The checker compares the check bits with the check bits generated from the primary outputs of the original circuit.

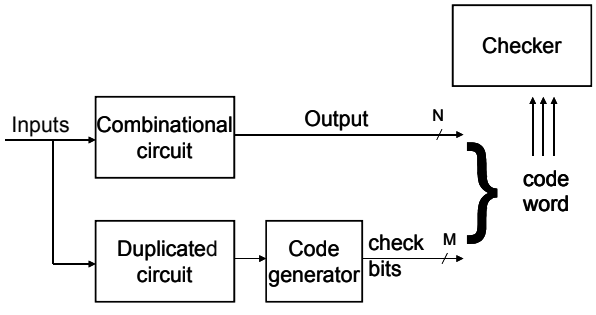

Fig. 1. Structure of TSC circuit

To satisfy the self-checking property the checker must have at least two outputs (Nicolaidis, 1998). The first output is used for the regular operation and the second for the error indication. This basic structure ensures that a circuit can be totally self-checking (TSC). Another condition that has to be satisfied is that the basic structure has to be self-testing and fault secure. The blocks "Duplicate circuit" and "Code generator" can be minimized and, after that, the resulting design of them together can be smaller than the original tested circuit.

3.1 Adopted fault model

In all of our experiments FPGA circuits are used. The circuit implemented in an FPGA consists of individual memory elements (LUTs - look up tables). In Fig. 2 we can see 3 gates mapped into a LUT.

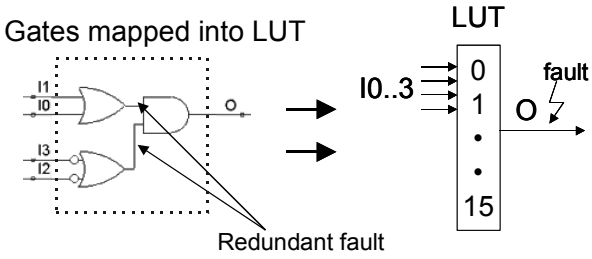

Fig. 2. Fault model

The original circuit has two inner nets. The original set of the test vectors covers all the stuck-at faults occurring in these inner nets. For the LUT these test vectors are redundant. For circuits realized by LUTs the change (a defect) in the memory leads to a single event upset (SEU) at the primary output of the LUT. Therefore we can use the stuck-at fault model in our experiments to detect SEU, while some of the detected faults will be redundant.

The used fault model is described by a simple example in Fig. 3. We have used only one LUT, for the simplicity. The LUT realizes a circuit containing 3 gates. Primary inputs I0 to I3 are the same as the address inputs for the LUT. When this address is selected, its content is propagated to the output.

We assume the following situation: The content of this LUT can be changed, e.g., by an electromagnetic interference, cross-talk or alpha particles. The appropriate memory cell is set to one and the wrong value is propagated to the output. It means that the realized function is changed and output behaves as a single event upset. By this example we can say that a change of any LUT cell leads to a stuck-at fault on the output. This fault is observed only if the faulty cell is selected. This applies to circuits constructed from elementary gates as well. Some fault can be masked and does not necessarily lead to an erroneous output. Due to the fact that some faults are masked, the possibility of their occurrence may be situated in time when logic is used. For example if one bit of a LUT is changed, the erroneous output will appear, while the appropriate bit in the LUT is selected by the address decoder.

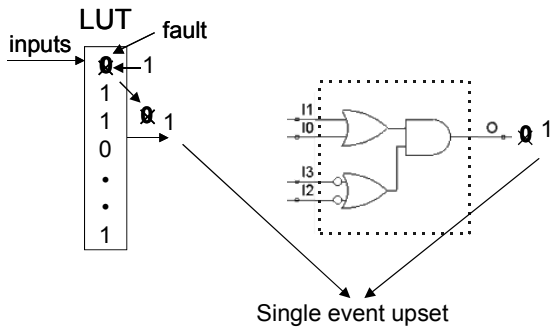

Fig. 3. Fault Model - Example

3.2 Principle of Hamming code generator

There are many possibilities how to generate the checking bits. A single even parity code is the simplest code that may be used to get a code word at the output of the combinational circuit. This parity generator performs XOR over all primary outputs. In most cases single even parity code is not appropriate to ensure the TSC goal.

Another error code is a Hamming code that is in essence based on single parity code (multi parity code). The Hamming code is defined by its generating matrix. For simplicity we use the matrix containing the unity sub-matrix at the left side. The generating matrix of the Hamming code (15, 11) is shown in Fig. 4. The values a_{ij} have to be defined.

$$G = \left(\begin{array}{cccc|cccc} 1 & 0 & \cdots & 0 & a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ 0 & 1 & \cdots & 0 & a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & a_{11,1} & a_{11,2} & a_{11,3} & a_{11,4} \end{array}\right)$$

Fig. 4. Generating matrix for Hamming code (15, 11)

When more complex Hamming code is used, more values have to be defined. The number of outputs o_i used for checking bits determines the appropriate code. E.g. the circuit alu1 (see MCNC) that has 8 outputs requires at least the Hamming code (15, 11). In this case we use 8 data bits and 4 checking bits. The definition of the values a_{ik} is also important. Now we present a method how to generate values a_{ik}. Let us mention the Hamming code (15, 11) that has 4 checking bits. In our case (alu1) we have only 8 bits. The reduced Hamming matrix must be used.

After the reduction the sub-matrix has only 8 rows and 4 columns. We can define 8 of 4-bit vectors or 4 of 8-bit vectors. The second case will be used here. The method searching for erroneous output is similar to the binary search method. The first vector is composed of log. 1s only. And the last vector is composed of log. 1s on the odd places and log. 0s on the even places. Every vector except the first one contains the same number of 1s and the same number of 0s. An example of the possible content of the right side sub-matrix is shown in Fig. 5.

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Fig. 5. Generating right side of matrix

The number of vectors in the set is the same as the number of rows in the appropriate Hamming matrix. How to generate parity output for checking bit x_k is described by equation 1

$$x_k = a_{1k}o_1 \oplus a_{2k}\, o_2 \oplus \ldots \oplus a_{mk}o_m, \qquad (1)$$

where $o_1 \ldots o_m$ are the primary outputs of the original circuit.

3.3 Two ways of area overhead minimization

The newly proposed simplification method consists in converting the duplicate circuit into a two-level network, for which the check-bits are generated. Then the outputs of the circuit are reduced to these check-bits only; the original outputs can be omitted. After that, a multi-level network is synthesized for this circuit. This notion enables us to significantly reduce the resulting logic.

The benchmarks, which are used in this paper, are described by a two-level network in order to compute the quality of the code. Two ways to generate check bits were used to compare the possibilities of reducing the check bits generator. The first way is based on a modification of the circuit described by a two-level network. The area of the check bits generator contributes significantly to the total area of the TSC circuit. As an example we consider a circuit with 3 inputs (c, b and a) and 2 outputs (f and e). The check bits generator uses the odd parity code to generate check bits. In our example we have only one check bit x.

Our example is shown in Table 1. The output x was calculated from the outputs e and f. At this time we have to generate the minimal form of the equation. We can achieve the minimal form using methods like the Karnaugh map or Quine-McCluskey. After the minimization we obtain three equations, one per every output (f, e and x), where x means an odd parity of the outputs f and e. If we want to know if the odd parity covers all faults in our example of simple combinational circuit, we have to generate the minimal test set and simulate all faults on every net in this circuit.

Table. 1. Example of parity generator

c b a	f e	x
0 0 0	0 1	0
0 0 1	1 0	0
0 1 0	1 0	0
0 1 1	1 0	0
1 0 0	0 1	0
1 0 1	0 1	0
1 1 0	1 1	1
1 1 1	0 0	1

The final equations are:

$$e = bc + a(b + c) \quad (2)$$
$$f = ab + c(a + b) \quad (3)$$
$$x = bc \quad (4)$$

The second way is based on a modification of multi-level network. The parity bits are added as a tree composed of XOR gates. The maximal area of the parity generator can be calculated as a sum of the original circuit and the size of the XOR tree.

3.4 Software for experimental evaluation

Fig. 6. describes how the test is performed for every detecting code. We have used the MCNC benchmarks (see MCNC) in our experiments. These benchmarks are described by a truth table. To generate the output parity bits, all the output values have to be defined for each particular input vector. Unfortunately, it is not so in the benchmark definition files. Only several output values are specified for each multi-dimensional input vector, the rest are assigned as don't cares; they are left to be specified by another term. Thus, to be able to compute the parity bits, we have to split the intersecting terms, so that all the terms in the truth table are disjoint.

In the next step the original primary outputs are replaced by parity bits. Two different error codes were used to calculate output parity bits (single even parity code and Hamming code). Another tool was used in the case where the original circuit was modified in multilevel logic. This tool is described in (Kubalík, 2003). Two circuits generated in the first step (original circuit and parity circuit) are processed separately to avoid sharing any part of circuit. Every part is minimized by the Espresso tool (Brayton, 1984). The final area overhead depends on the software that was used in this step. Many tools were used to reach a small area of the parity bits generator. Only Espresso was used to minimize the final area. In this step the area overhead is known, but we can decide if the fault coverage is sufficient.

In the next step the "pla" format is converted into the "bench" format. The "bench" format was used due to the fact that the tool, which generates the exhaustive test set uses this format. An exhaustive test set has 2^n patterns and we used it to evaluate TSC goals.
Another conversion tool is used to generate two VHDL codes and the top level. Top level is used for incorporating original and parity circuit generator. In the next step the synthesis process is performed by Synplify Pro. The constraints properties set during the synthesis process express an area overhead and a fault coverage. If the maximal frequency is too high, hidden faults occur during the fault simulation. The hidden faults are caused by the circuit duplication. The size of the area overhead is obtained from the synthesis process. The final netlist is generated by the Leonardo Spectrum software. The fault coverage was obtained by simulation using our software.

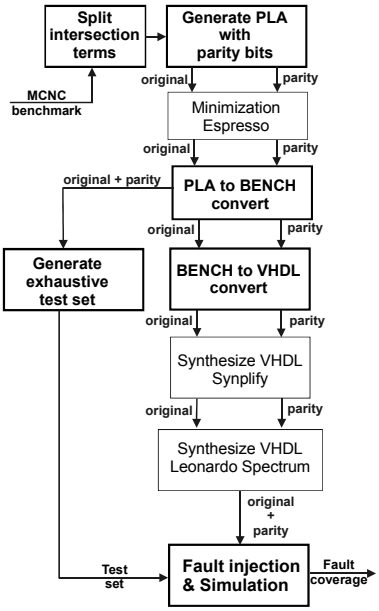

Fig. 6. Design scheduling of self-checking circuit

3.5 Software solution description

To evaluate the area overhead and fault coverage special tools had to be developed. In addition to some commercial tools such as Leonardo Spectrum and Synplify we have used format converting tools, ATPG (Automatic Test Pattern Generator) tools, parity circuit generator tools and simulation tools.

At first, the area minimization and term splitting is preformed for original circuit by BOOM (Hlavicka, 2001). Hamming code generator (or single parity generator) is generated by the second software. These two circuits are minimized again with Espresso. Next two tools convert the two-level format into a multi-level format. The first one converts a "pla" file to "bench" and the second one "bench" to VHDL. The second software is used for generating the final circuit in the "bench" format due to their further usage in exhaustive test set generator. The format converting software and parity generator software was written in Microsoft Visual C++. The netlist fault simulator was written in Java. The parser source

code was used for parsing the netlist that is generated by two commercial tools described above.

4 EXPERIMENTAL RESULTS

The combinational MCNC benchmarks (MCNC) were used for all the experiments. These benchmarks are based on real circuits used in large designs. Due to the fact that the whole circuit will be used for reconfiguration in FPGA, only small circuits were used. Real designs having a large structure must by partitioned into several smaller parts. For large circuits the process of the area minimization and fault simulation takes a long time. This disadvantage prevents us to examine more methods of designing the check bits generator.

Table 2 Description of tested benchmarks

Circuit	Inputs	Outputs
alu1	12	8
apla	10	12
b11	8	31
br1	12	8
al2	16	47
alu2	10	8
alu3	10	8
c17	5	2

The evaluated area and fault coverage depend on the circuit properties such as number of inputs, outputs and gates. Experimental results show that a more important property is the structure of the used circuit. Two basic properties are described in Table 2.

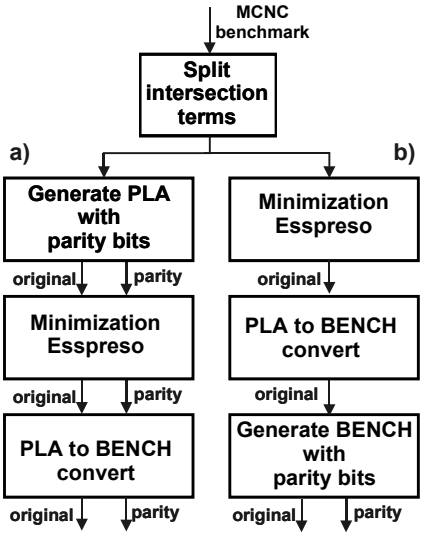

Fig. 7. Two different flows of creating parity generator

In the first set of experiments our goal was to obtain a hundred percent fault coverage, while we measured the area overhead. In this case the maximum of the parity bits was used.

This task was divided into two experiments (Fig. 7). In the first experiment the two-level network was being modified (Fig. 7a). The results are shown in Table 3.

Table 3 Hamming code – PLA

Circuit	Parity nets	Original [LUT]	Parity [LUT]	Overhead [%]	Fault coverage
alu1	4	8	84	1050	100
apla	5	45	105	233	100
b11	6	38	38	100	100
br1	4	50	59	118	97,8
al2	7	51	54	106	100
alu2	4	30	127	423	100
alu3	4	28	94	336	100
c17	2	2	3	150	100

Table 4 Hamming code – XOR

Circuit	Parity nets	Original [LUT]	Parity [LUT]	Overhead [%]	Fault coverage
alu1	4	8	13	163	100
apla	5	45	114	253	100
b11	6	38	73	192	100
br1	4	50	85	170	98,3
al2	7	52	109	210	100
alu2	4	30	52	173	100
alu3	4	28	44	157	100
c17	2	2	3	150	100

The 100% fault coverage was fulfilled in 7 cases. The area overhead in many cases exceeds 100%. It means that the cost of a hundred percent fault coverage is too high. In these cases the TSC goal is satisfied. Then, we have used an old method where the original circuit in multi-level network is being modified by additional XOR logic (Fig. 7b). (Kubalík, 2003).

The results obtained from this experiment are shown in Table 4. A 100% fault coverage condition was fulfilled in the same cases as in the first experiment but the overhead is in some cases smaller. In the second set of experiments we have tried to obtain a small area overhead and the fault coverage was measured. In this case the minimum of the parity bits is used (single even parity).

The experiments are divided into two groups a) b), Fig. 7. The procedure is the same as described above. In the first experiment the two-level network of the original circuit was modified (Fig. 7a). The results are shown in Table 5. The 100% fault coverage is reached in two cases, however an area overhead is smaller in five cases. In the last experiment we have modified the circuit described by a multilevel network (Fig. 7b). In many cases the area overhead is

higher than 100%, however, the fault coverage did not increase, Table 6.

Table 5. Single even parity - PLA

Circuit	Parity nets	Original [LUT]	Parity [LUT]	Overhead [%]	Fault coverage
alu1	1	8	271	3388	100
apla	1	46	23	50	82,6
b11	1	37	3	8	77,3
br1	1	54	10	19	62,1
al2	1	52	4	8	91,9
alu2	1	29	47	162	92,5
alu3	1	26	32	123	92
c17	1	2	2	100	100

Table 6. Single even parity - XOR

Circuit	Parity nets	Original [LUT]	Parity [LUT]	Overhead [%]	Fault coverage
alu1	1	8	10	125	100
apla	1	46	56	122	88,2
b11	1	37	36	97	86,2
br1	1	54	61	113	78,9
al2	1	52	23	44	93,5
alu2	1	29	44	152	91,1
alu3	1	26	39	150	91,6
c17	1	2	2	100	100

5. CONCLUSION AND FUTURE WORK

All of our experiments say that TSC goals can be reached for the whole design, including the checking parts. It is achieved by using more redundant outputs generated by special codes. The final area overhead significantly depends on the number of inputs and outputs, parity bits and the structure of a tested benchmark. In cases where one parity bit is used, the final area overhead is smaller than 100%, but TSC goals are not fulfilled. In cases where TSC goals are met the area overhead is greater then 100%.

All of our experiments apply to combinational circuits only. But many circuits in real designs are composed of sequential parts, too. However such circuits can be divided into simple combinational parts separated by flip-flops. The finite state machine can be divided into two parts: the first part covers combinational logic from inputs to flip-flops (with feedback), the second one covers the combinational logic from flip-flops to outputs (with the nets that are connected directly from the input to the output). Therefore the restriction to the combinational circuits does not reduce the quality of our methods and experimental results. In the future we have to discover more precise relations between real FPGA defects and the used fault models. Also the appropriate decomposition of the designed circuit is under our intensive research.

ACKNOWLEDGEMENT

This research has been in part supported by the GA102/04/2137 grant, CTU0408913 grant and MSM 212300014 research program.

REFERENCES

Abramovici, M., C. Stroud, S. Wijesuriya, C. Hamilton, and V. Verma (1999). Using Roving STARs for On-Line Testing and Diagnosis of FPGAs in Fault-Tolerant Applications, *Proc. IEEE Intn'l. Test Conf.*, pp. 973-982.

Bolchini, C., F. Salice and D. Sciuto (2002). Designing Self-Checking FPGAs through Error Detection Codes, *17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'02)*, pp. 60, Canada.

Brayton, R. K. et al. (1984). *Logic minimization algorithms for VLSI synthesis*, Boston, MA, Kluwer Academic Publishers, 192 pp.

Bushnell, M. L. and V. D. Agrawal (2000). *Essentials of Electronic Testing*. Kluwer Academic Publisher, London.

Drineas, P., Y. Makris (2003). Concurrent Fault Detection in Random Combinational Logic, In: *Proceedings of the IEEE International.* Symposium on Quality Electronic Design (ISQED), pp. 425-430.

Hlavička, J. and P. Fišer (2001): BOOM - a Heuristic Boolean Minimizer. *Proc. International Conference on Computer-Aided Design ICCAD 2001*, San Jose, California (USA), pp. 439-442

Kubalík, P. and H. Kubátová (2003). Design of Self Checking Circuits Based on FPGA. In: *Proc. of 15th International Conf. on Microelectronics*, pp. 378-381. Cairo, Cairo University.

MCNC - ftp://ic.eecs.berkeley.edu

Mitra, S., and E. J. McCluskey (2000a). Which Concurrent Error Detection Scheme To Choose? *Proc. International Test Conf.*, pp. 985-994.

Mitra, S., and E. J. McCluskey (2000b). *Diversity Techniques for Concurrent Error Detection*, Center for Reliable Computing, Dept. of Electrical Engineering and Computer Science Stanford University, Technical Report 00-7.

Mohanram, K., E. S. Sogomonyan, M. Gössel, N. A. Touba (2003). Synthesis of Low-Cost Parity-Based Partially Self-Cheking Circuits, *Proceeding of the 9th IEEE International On-Line Testing Symposium (IOLTS'03)*, pp. 35.

Nicolaidis, M. and Y. Zorian (1998). *On-Line Testing for VLSI* - A Compendium of Approaches." On-Line Testing for VLSI, Kluwer Academic Publisher, London.

Stroud, Ch. E. (2002). *A Designer's Guide to Built-In Self-Test*. Kluwer Academic Publisher, London.

Xilinx (2003). XAPP 151 (v1.6), *Virtex Series Configuration Architecture User Guide*