Analyzing the Test Generation Problem for an Application-Oriented Test of FPGAs

M. Renovell, J.M. Portal, P. Faure
LIRMM-UM2
161 Rue Ada
34392 Montpellier France
renovell@lirmm.fr
Tel (33)467418523

J. Figueras
UPC
647 Av Diagonal
Barcelona Spain
figueras@eel.upc.es
Tel (34)34016603

Y. Zorian
Logic Vision Inc.
101 Metro Drive
San Jose CA 95110 USA
zorian@lvision.com
Tel (1)4084530146

Abstract

The objective of this paper is to generate a Application-Oriented Test Procedure to be used by a FPGA user in a given application. General definitions concerning the specific problem of testing RAM-based FPGAs are first given such as the important concept of 'AC-non-redundant fault'. Using a set of circuits implemented on a XILINX 4000E, it is shown that a classical test pattern generation performed on the circuit netlist gives a low AC-non-redundant fault coverage and it is pointed out that test pattern generation performed on a FPGA representation is required. It is then demonstrated that test pattern generation performed on the FPGA representation can be significantly accelerated by removing most of the AC-redundant faults. Finally, a technique is proposed to even more accelerate the test pattern generation process by using a reduced FPGA description.

1. Introduction

Field Programmable Gate Arrays (FPGAs) combine the flexibility of mask programmable gate arrays with the convenience of field programmability [1-12]. There are many FPGA types but one very important is the static-RAM based FPGA architecture. In such a programmable circuit, a matrix of logic modules and interconnection elements can be configured in the field to implement a desired function. Recently, testing such flexible circuits from a manufacturer point of view has received more attention in the test community [2-11].

Testing for FPGAs, is a difficult and important problem. Assuming for example we have to generate a test sequence for the XC4025 of Xilinx. The different test tools used for this purpose as Automatic Test Pattern Generator or fault simulator, have to handle a sequential circuit with about 2560 flip-flops, 256 I/O pins, a mix of logic modules and 32,768 static RAM bits. Concerning the logic modules, they are not composed of gates but mainly composed of transistor based multiplexer implying a problem of circuit representation. Concerning the fault models, memory oriented models have to be used for the RAM cells, gate oriented models for the logic modules and transistor oriented models for the interconnection modules.

From the test point of view, it is now widely admitted that FPGAs can not be considered as classical digital ASICs. Classical test approaches fail when applied on FPGA. In the recent published works, different test aspects are considered: boolean testing of FPGA [2,3,5-11], BIST for FPGA [2,3], Iddq testing of FPGA [4], diagnosis of FPGA [6]. Because of the complexity of FPGA testing, usually each paper targets a specific FPGA part: the interconnect in [3,5,7], the logic cells in [2,4,8], the memory cells in [9,10,11], the logic-interconnect interface. Indeed, FPGAs appear as very complex circuits and these works use a classical divide and conquer approach.

It must be pointed out that all the previously mentioned works try to generate a FPGA test to be used after manufacturing, their objective is to generate a Manufacturing-Oriented Test Procedure. The objective of this paper is to generate a FPGA test to be used by a FPGA user in a given application, our objective is to generate an *Application-Oriented Test Procedure*. To our knowledge, this is one of the first papers dedicated to the generation of an external Application-Oriented Test Procedure for FPGAs. As mentioned in the above

paragraph, FPGAs are large and complex circuits and so, this paper reasonably can not target the whole FPGA but rather a part of the FPGA. So, we defined more precisely the objective of this paper as the generation of an Application-Oriented Test Procedure for the logic cells of a FPGA.

This paper is organized in the following way. Section 2 proposes general definitions concerning the specific problem of testing RAM-based FPGAs. Among other important points, this section introduces the important concept of 'AC-non-redundant fault'. Using a set of example of small combinatorial circuits implemented on a XILINX 4000E, section 3 shows that a classical test pattern generation performed on the circuit netlist gives a low AC-non-redundant fault coverage and points out that a test pattern generation performed on a FPGA description is required but very time consuming. Section 4 demonstrates that test pattern generation performed on the FPGA representation can be significantly accelerated by removing most of the AC-redundant faults. Section 5 proposes to even more accelerate the test pattern generation process by using a reduced FPGA description. Finally, section 6 concludes.

2. FPGA Test Definitions

The purpose of this section is to give general definitions about the specific problem of generating an Application-Oriented FPGA test procedure.

A RAM-based FPGA exhibits two different types of input: the 'Operation' Inputs and the 'Configuration' Inputs. The 'Operation' Inputs are used during the circuit normal operation to apply the input vectors $IV_1...IV_n$ and the 'Configuration' inputs are used before the circuit normal operation to configure the FPGA. A given FPGA 'Configuration' consists in a long bit stream serially entered in the circuit.

At this point, it must be pointed out the user does not need to test the complete FPGA. He is only interested in testing the part of the FPGA used for his specific application. In this case, the configuration used for test purpose correspond to the configuration used in his application. This application-oriented configuration is simply called « Application Configuration » and denoted AC. These remarks lead to the following definitions.

Definition 1 *An Application-Oriented test procedure for FPGA is represented as an Application Configuration AC and its associated Test Sequence TS:*

$$\boldsymbol{AOTP=\{(AC,TS)\}} \qquad with\ TS=(IV^i_1,IV^i_2,..)$$

For the AOTP, the configuration is fixed by the application and the problem consists in defining the Test Sequences TS associated to this configurations AC. Of course, the detectability of a given fault depends on the AC circuit configuration. Some redundant and non-redundant faults are described in the following sections, but as an example, it is quite obvious to imagine that a fault located in a portion of the FPGA not used by a given configuration is undetectable and so redundant in this configuration.

Definition 2 *Considering a FPGA Configuration C^i, a fault is said **C^i-redundant** (resp. **C^i-nonredundant**) if the fault is redundant (resp. nonredundant) in this particular Configuration C^i.*

Obviously, definition 2 applies to an Application Configuration AC. We speak about AC-redundant fault (resp. AC-nonredundant). For the Application-Oriented Test Procedure, the Application Configurations **AC** is used and the problem consists in only generating the Test Sequence associated to this Application Configurations. So the quality metrics associated to the AOTP definition has to take into account the Test Pattern Generation process. It is now possible to propose a quality metrics for the Application-Oriented Test Procedure.

Definition 3 *The fault Coverage C^{AOTP} of an Application Oriented Test Procedure **AOTP** is equal to the number n_d of detected faults divided by the number n_{nr} of AC-nonredundant fault:* $\boldsymbol{C^{AOTP} = n_d / n_{nr}}$

3. FPGA TPG Dilemma

This section illustrates the problem of test pattern generation for an Application Configuration with simples examples. It is first shown that a trivial approach based on the application circuit netlist leads to very low fault coverage. It is then shown that a more efficient approach based on the FPGA netlist with the Application Configuration gives obviously better result but the corresponding CPU time can be prohibitive due to an excessive complexity.

In the context of FPGA implementation, test vectors computed by specific TPG tool with the circuit netlist and adequate fault models do not work. It must be clear that the circuit netlist used in the design phase before implementation into the FPGA, does not contain any structural information on the final physical FPGA. The following experiment can easily illustrate this point. The circuits numbered 1 to 9 in the first column of **table 1** are small combinatorial circuit coming from benchmark circuits. The circuits they come from are indicated in column 2. Only fraction of benchmarks have been used in order to have small circuits. Indeed, there is presently no

tool implementing the proposed approach and so, all the manipulations required for this demonstration are made manually.

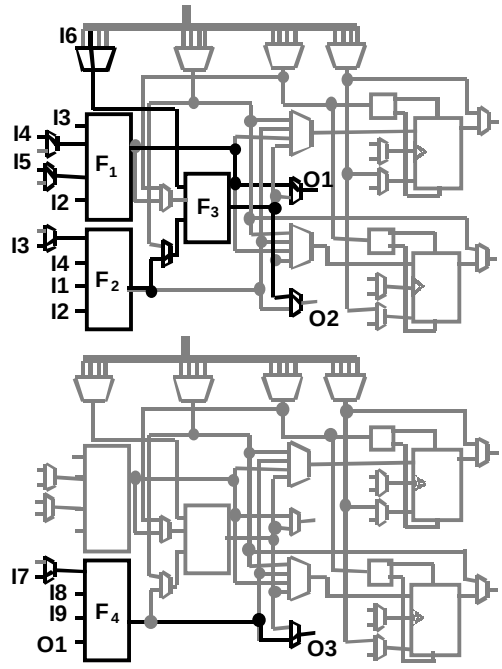
For each circuit, we use the circuit netlist to generate a test sequence using a classical commercial ATPG with the stuck-at fault model. Remember that the stuck-at fault coverage is in each case equal to 100%. Then the benchmark circuit is mapped into a XILINX 4000E FPGA using the XILINX tools (XILINX Fondation Series Tools). The result of the mapping is a configuration of the FPGA that is precisely the Application-Configuration discussed in the previous section. It is now possible to create a logic description of the FPGA including logic modules such as look-up tables, multiplexers, flip-flops and so on. As commented in the introduction, this paper is limited to the analysis of the logic cells and so the interconnect elements such as switch matrices are not represented. In the FPGA logic description, the configuration inputs of the logic modules are fixed and set to the value corresponding to the Application-Configuration.

#	Circuit Issued From	Sequence generated with Circuit netlist	Sequence generated with ALC-FPGA descr.
1	c17	83.81%	100%
2	s344	81.87%	100%
3	s382	85.48%	100%
4	s208	90.92%	100%
5	s382	87.69%	100%
6	s382	85.39%	100%
7	s208	91.42%	100%
8	s349	66.42%	100%
9	s298	90.77%	100%

Table 1: C^{AOTP} for different test sequences

According to the definitions given in section 2, what we have to detect is the set of AC-nonredundant fault, i.e. the set of stuck-at fault in the logic FPGA description that are not redundant when the Application-Configuration is used. Using now, the test sequence previously generated on the circuit netlist and the FPGA logic description with the Application-Configuration, we perform a classical stuck-at fault simulation and we note the corresponding fault coverage C^{AOTP}, i.e. the number of stuck-at fault detected divided by the total number of non-redundant stuck-at fault. The third column of **table 1** gives the fault coverage C^{AOTP} obtained for each benchmark circuit on the FPGA implementation. It is clear that the sequence generated on the circuit netlist does not give a satisfactory result on the real FPGA implementation. The resulting fault coverage varies from 66% to 91% which is absolutely not sufficient.

The previous experiment simply demonstrates that the circuit netlist can not be used for test pattern generation. Obviously, a FPGA logic description with the Application-Configuration can be used for test pattern generation and leads to better results. It must be noted that the FPGA description we use does not include the inactive logic cells. Indeed, when the circuit is implemented into the FPGA, i.e. in the matrix of logic cells, some of the logic cells are not used. These inactive logic cells are obviously not included in the FPGA description we use. Let call this description: **A**ctive-**L**ogic-**C**ell FPGA description (ALC). This ALC-FPGA description is illustrated in **figure 1** with circuit #1 issued from the c17 benchmark .



$F_1 = I4 \cdot I3 + \overline{I5} \cdot \overline{I2}$ $F_3 = \overline{I6} \cdot F_2$

$F_2 = I3 \cdot I4 \cdot \overline{I1} + \overline{I2} \cdot (\overline{I3} + \overline{I1})$ $F_4 = O1 \cdot I9 \cdot \overline{I8} \cdot \overline{I7}$

Figure 1: ALC-FPGA description of circuit #1

The last column of **table 1** gives, for each circuit of **table 1**, the fault coverage C^{AOTP} obtained with a test sequence generated directly on the ALC-FPGA description. Note that a 100% fault coverage is obtained in each case. This simple experiment demonstrates that a FPGA description must be used for test pattern generation when satisfactory fault coverage are required. But, it is clear that the FPGA description is by definition much more complex than the circuit netlist because of the inherent flexibility of the FPGA. This complexity leads to

higher CPU time for test pattern generation. **Table 2** gives the CPU time for a test pattern generation performed on the circuit description and the CPU time for a test pattern generation performed on the ALC-FPGA description. It is clear that the FPGA description requires much more time and this may lead to prohibitive CPU time for large circuits.

#	CPU time Circuit Netlist	CPU time ALC-FPGA
1	0.30s	0.97s
2	0.38s	1.80s
3	0.35s	1.09s
4	0.36s	3.71s
5	0.39s	6.16s
6	0.54s	12.55s
7	0.52s	35.83s
8	0.17s	75.84s
9	0.70s	87.46s

Table 2: CPU time for different descriptions

4. AC-redundant Fault

During test pattern generation on a FPGA description an important fraction of the time is waste due to a huge number of AC-redundant faults. It is interesting to note the following FPGA specificity. For an ASIC, the number of redundant faults is usually a very small fraction of the number of the non-redundant ones while it is the reverse situation for a FPGA configuration. **Table 3** indicates in column 2, the percentage of AC-redundant faults in the ALC-FPGA description. It can be observed that these AC-redundant faults are 2 to 3 times more numerous than the AC-nonredundant ones. To illustrate the effect of the AC-redundant fault on the ATPG CPU time, the third column of **table 3** gives the CPU time required on the ALC-FPGA description with the complete list of faults while the fourth column gives the CPU time required on the ALC-FPGA description with only the AC-nonredundant faults. The reduction factor given in the last column indicates reduction up to 93% may be obtained. It is clear at this point that, for efficiency purpose, test pattern generation must be performed on the FPGA logic description. But, to save CPU time, the AC-redundant faults must be identified and removed from the fault list prior to test pattern generation. For a FPGA logic description, we can define different classes of AC-redundant faults:

Class a) AC-redundant faults due to logical redundancy,
Class b) AC-redundant faults due to unused logic,
Class c) AC-redundant faults due to constant signal.

#	.AC-red. %	CPU time All faults	CPU time AC-nonred	Reduction %
1	44.76	0.97s	0.93s	-4,1
2	61.27	1.80s	0.94s	-47.77
3	61.69	1.09s	0.84s	-22.93
4	51.26	3.71s	3.03s	-18.32
5	57,46	6.16s	1.54s	-75.00
6	51.29	12.55s	5.88s	-53.14
7	56.20	35.83s	9.04s	-74.76
8	59.27	75.84s	11.47s	-84.87
9	67.28	87.46s	5.72s	-93.45

Table 3: CPU time with/without AC-redundant faults

In any kind of digital circuit some faults may be redundant due to some logical redundancy. In our case, it must be pointed out that these faults are redundant independently of the implementation. They are redundant in the FPGA implementation as well as in another type of implementation such as an ASIC implementation. They belong to the classical set of logically redundant fault that are very hard to identify a priori and so, will not be targeted here. The number of such redundant fault is usually very limited.

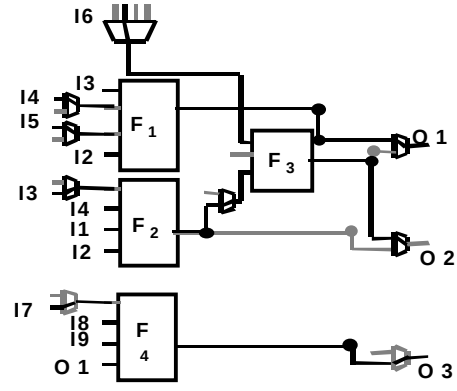


Figure 2: AM-FPGA description of circuit #1

The class 'b' fault affecting unused logic of the FPGA are redundant. **Figure 1** gives an illustration of the ALC-FPGA description of circuit #1 where the gray modules and gray lines are the ones not used in this configuration. Concerning first the gray modules, it is possible to remove all these unused modules from the description, removing de facto the corresponding redundant faults on these modules. The resulting description called **A**ctive-**M**odule-FPGA description is given in **figure 2**. Concerning now the remaining gray lines on the multiplexers, the stuck-at fault affecting these lines are redundant because the fault can not be propagated. Indeed, in a given FPGA configuration, the multiplexers

configuration inputs are set to constant values. So, these faults are removed a priori from the fault list.

In the AM-FPGA description of **figure 2**, the LUT configuration inputs and the multiplexer configuration inputs are set to constant values. This is inherent to the FPGA flexibility. If we consider now a given configuration input set, for example, to '0', the corresponding stuck-at-0 fault is class 'c' redundant. All these fault corresponding to constant signals are easily identified and removed from the fault list.

In **table 3** column 4, all the AC-redundant faults belonging to class 'b' and 'c' and described above, have been removed a priori from the fault list. The indicated CPU times consequently illustrates the efficiency of the proposed approach. Note that the reduction factor is extremely high for the largest circuits.

5. ATPG acceleration

When AC-redundant faults are removed a priori from the list, test pattern generation is considerably accelerated as discussed in the previous section. However it is still possible to accelerate the test pattern generation process by identifying a priori equivalent faults or by simplifying the FPGA description and by exploiting LUT properties.

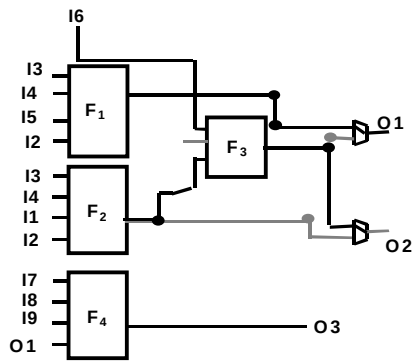


Figure 3: SIM-FPGA description of circuit #1

#	CPU time AM-FPGA	CPU time SIM-FPGA	Reductio n %
1	0.93s	0.75s	-19.35
2	0.94s	0.82s	-12.76
3	0.84s	0.83s	-01.20
4	3.03s	2.95s	-02.64
5	1.54s	1.24s	-19.48
6	5.88s	5.02s	-14.62
7	9.04s	3.10s	-65.70
8	11.47s	7.08s	-38.27
9	5.72s	3.94s	-31.11

Table 4: CPU time on the Simplified Description

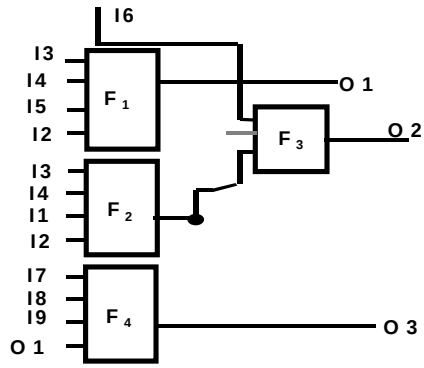


Figure 4: RED-FPGA description of circuit #1

#	CPU Time SIM-FPGA	CPU Time RED-FPGA	Reductio n %	Coverage AM-FPGA
1	0.75s	0.75s	-00.00	100%
2	0.82s	0.66s	-19.51	100%
3	0.83s	0.83s	-00.00	100%
4	2.95s	2.95s	-00.00	100%
5	1.24s	1.14s	-08.06	100%
6	5.02s	4.68s	-06.77	100%
7	3.10s	3.10s	-00.00	100%
8	7.08s	5.58s	-21.18	100%
9	3.94s	3.06s	-22.33	100%

Table 5: CPU time on the Reduced Description

The AM-FPGA description includes only active modules. However, we observe in **figure 2** that some multiplexers have all the side inputs set to constant value as for example the multiplexer driven by Input I6. For these multiplexers, all the remaining AC-nonredundant stuck-at faults including faults affecting the configuration inputs, are equivalent to the multiplexer output stuck-at fault. This means that these multiplexers can be removed from the description during test pattern generation. The stuck-at of the remaining logic node covers all the considered multiplexer related stuck-at faults. By remaining these multiplexers from the AM-FPGA description we obtain a new description called SIMplified-FPGA description illustrated in **figure 3**. To validate this property, the test sequences generated on the SIM-FPGA descriptions have been applied to the AM-FPGA description and a 100% stuck-at fault coverage has been obtained. **Table 4** shows the corresponding CPU time and the reduction factor given in the last column indicates that reduction up to 65% may be obtained.

Finally, the CPU time can still be reduced by proposing an heuristic. In the SIM-FPGA of **figure 3**, all the remaining multiplexers have a side input not used but

connected to another logic node of the circuit as for example the multiplexer driving node O2: its used input is connected to node F3 while its unused side input is connected to node F2. The fault affecting the configuration inputs of this multiplexer may disconnect node F3 and connect the multiplexer output to node F2. Of course this type of fault must be detected by setting node F2 and F3 to different values. The heuristic we use here for this type of fault is the following: the test of LUT F2 and F3 implementing different functions generally imply different values for node F2 and F3. And so the test of LUT F2 and F3 covers this type of fault. There is no need for a specific test pattern generation for these fault and so we can consequently remove these remaining multiplexers. By remaining these multiplexers from the SIM-FPGA description we obtain a REDuced-FPGA description illustrated in **figure 4**. To validate this property, the test sequences generated on the RED-FPGA descriptions have been applied to the AM-FPGA description and a 100% stuck-at fault coverage has been obtained for each circuit. **Table 5** shows the CPU time for a test pattern generation performed on the RED-FPGA description. The reduction factor given in the last column indicates that reduction up to 22% may still be obtained.

6. Conclusion

This paper studies the problem of generating an Application-Oriented Test Procedure to be used by a FPGA user in a given application. General definitions concerning the specific problem of testing RAM-based FPGAs have been given including the important concept of 'AC-non-redundant fault'.

#	CPU time ALC-FPGA	CPU Time RED-FPGA	Reduction %
1	0.97s	0.75s	-22.68
2	1.80s	0.66s	-63.33
3	1.09s	0.83s	-23.85
4	3.71s	2.95s	-20.48
5	6.16s	1.14s	-81.49
6	12.55s	4.68s	-62.71
7	35.83s	3.10s	-91.35
8	75.84s	5.58s	-92.64
9	87.46s	3.06s	-96.50

Table 6: Global CPU time reduction

Using a set of circuits implemented on a XILINX 4000, we have shown that a test pattern generation performed on a FPGA representation is required but very time consuming. It is then demonstrated that test pattern generation performed on the FPGA representation can be significantly accelerated by removing a priori most of the AC-redundant faults from the fault list. Finally, an heuristic is proposed to even more accelerate the test pattern generation process by using a reduced FPGA description for test pattern generation. The global CPU time reduction obtained by using all the proposed techniques is given in **table 6**. It is clear that an important reduction is obtained in each case with better results on the largest circuits.

7. References

[1] S.M. Trimberger (ed), « Field-Programmable Gate Array Technology », Kluwer Academic Publishers, 1994.

[2] C. Stroud, P. Chen, S. Konala, M. Abramovici, "Evaluation of FPGA Ressources for Built-In Self Test of Programmable Logic Blocks", Proc. of 4th ACM/SIGDA Int. Symposium on FPGAs, pp. 107-113, 1996.

[3] C. Stroud, S. Wijesuraya, C. Hamilton, M. Abramovici, "Built-In Self Test of FPGA Interconnect", International Test Conference, pp. 404-411, Oct. 18-23, 1998, Washington, USA.

[4] L. Zhao, D.M.H. Walker, F. Lombardi, "Detection of Bridging Faults in Logic Resources of Configurable FPGAs Using IDDQ", International Test Conference, pp. 1037-1046, Oct. 18-23, 1998, Washington, USA.

[5] M. Renovell, J. Figueras and Y. Zorian, «Test of RAM-Based FPGA: Methodology and Application to the Interconnect», 15th IEEE VLSI Test Symposium, pp. 230-237, Monterey, CA, USA, May 1997.

[6] F. Lombardi, D. Ashen, X.T. Chen, W.K. Huang «Diagnosing Programmable Interconnect Systems for FPGAs», FPGA'96, pp. 100-106, Monterey, USA, 1996.

[7] H. Michinishi, T. Yokohira, T. Okamoto, T. Inoue, H. Fujiwara, "A Test Methodology for Interconnect Structures of LUT-based FPGAs", IEEE 5th Asian Test Symposium, pp. 68-74, November 1996.

[8] M. Renovell, J.M. Portal, J. Figueras and Y. Zorian, «Minimizing the number of Test Configurations for different FPGA Families», IEEE 8th Asian Test Symposium, November 1999, Shangai, China.

[9] M. Renovell, J.M. Portal, J. Figueras and Y. Zorian, «SRAM-based FPGA: Testing the LUT/RAM Modules», IEEE International Test Conference, pp. 1102-1111, Washington, DC, USA, Oct. 18-23, 1998.

[10] H. Michinishi, T. Yokohira, T. Okamoto, T. Inoue, H. Fujiwara, "Testing for the Programming Circuits of LUT-based FPGAs", IEEE 6th Asian Test Symposium, pp. 242-247, November 1997.

[11] W.K. Huang, F.J. Meyer, N. Park and F. Lombardi, «Testing Memory Modules in SRAM-based Configurable FPGAs», IEEE International Workshop on Memory Technology, Design and Test, August, 1997.

[12] Xilinx, «The Programmable Logic Data Book », San Jose, USA, 1994.