# Configuration-Specific Test Pattern Extraction for Field Programmable Gate Arrays

F. Ferrandi F. Fummi L. Pozzi M.G. Sami
Dip. di Elettronica e Informazione
Politecnico di Milano
Milano, ITALY 20133

**Abstract**

*The aim of this paper is to present a methodology for extracting configuration-specific test patterns for FPGA cells, from the set of sequences that test all stuck-at-faults for the unconfigured cell. This is achieved through the construction of an automaton that recognises test sequences for all faults, followed by the extraction of a second automaton that recognises only the non-redundant faults with respect to a given configuration. Since structural information is not needed for sequence extraction, this methodology provides the user with a structural fault model while granting protection of Intellectual Property.*

## 1 Introduction

The problem of testing programmable devices can be considered a particular case of the more general issue of testing Application Specific ICs. Indeed, FPGAs are ASICs with the more specific characteristic of reconfigurability. Due to this feature, the classical distinction of 'manufacturer testing' vs. 'user testing' takes a new meaning. On the one hand, end-of-production testing must grant *total* fault coverage, independently of specific configurations mapped onto the FPGA: thus it should take into account all possible configurations. On the other hand, the user is interested in testing the *programmed* FPGA (possibly even already inserted in a larger system) so that only the specific configurations mapped onto the chip need to be taken into account. To this end, as noticed in [5], the input pins of a reconfigurable device can be divided into two categories: normal inputs and configuration inputs. When generating end-of-production test patterns, the manufacturer has no constraints on the value of the configuration inputs; rather, testing shall exercise all possible configurations. On the contrary, as far as the user is concerned, testing is constrained to patterns with configuration inputs corresponding to the particular functions mapped onto the programmed device. Moreover, the resulting (programmed) circuit is a subset of the unconfigured one and will therefore

1063-6722/97 $10.00 © 1997 IEEE

show a high degree of redundancy due to gates that are not used in the specific configuration. It is therefore clear that the issue of testing FPGAs should involve different methodologies depending on whether the task is carried out by the manufacturer or by the user.

A general problem for user's testing has always been due to the lack of detailed structural information, because of Intellectual Property issues, which easily leads to unsatisfactory test pattern generation. It would then be of great interest if the manufacturer could allow the user to extract from a set of test sequences generated for the unconfigured cell, the subset of sequences needed to test the cell when *one* particular function (i.e. configuration) is mapped onto it, in such a way as to protect information on the internal structure of the cell itself.

It is only recently that the problem of testing FPGA cells has been taken into account. One previous work [5] has appreciated the difference of the manufacturer versus user approaches. In this respect it is related to our investigation, although it does not focus on the logic cell of the FPGA, but on the interconnect. In [8] the design for testability issue is tackled, and a mapping philosophy is suggested that allows to reach a testable configured device. The authors of [7] address the testing of unprogrammed and programmed logic circuits in FPGAs, and other authors have investigated FPGA testing in general, but all these works [6, 1] address a feature of the problem different from the one we focused on. To the best of our knowledge, no one has previously investigated the task of extracting test sequences for one particular configuration of an FPGA cell starting from the set of sequences that test the same unprogrammed cell. This is a novel issue in the field of testing research.

We propose a methodology that allows the user to generate test patterns for a specific configuration of an FPGA cell, through *extraction* from the set of sequences constructed for all possible configurations of the cell. This is achieved through three phases:

- The construction of an automaton, $GAF$, that recognises *all* the sequences for testing all faults of the cell.
- The extraction of the automaton $CGAF$, related to the specific configuration $C$.
- The traversal of such automaton to produce the final test pattern through concatenation and overlapping of the recognised sequences.

Note that this allows the users to obtain all test sequences for the programmed cell: this leads in general to a redundant set of sequences. Such redundancy can constitute a degree of freedom in detecting the 'best' minimum test set in relation to propagation and justification to primary inputs and outputs (from this point of view, the chip level testability aspects considered in [8] complement our cell level test generation).

The rest of this paper is organised as follows. Section 2 presents the notation that will be used throughout the exposition of the work and shows definitions regarding FPGA cell testing and test

sequence manipulation, while Section 3 presents the proposed technique. Experimental results obtained by application of the methodology to an FGPA cell are also shown. An important point concerns the protection of the manufacturer's property circuit information; this is tackled in section 4. Finally, section 5 is devoted to conclusions and directions for future work. In particular, use of cell-specific test patterns to create array level test patterns is outlined.

## 2 Definitions and Reference Methodology

We first present the notation related to FPGA cell testing; the symbolic entities used in the test sequence generation methodology are then described.

### 2.1 Cell Testing

Consistently with [5], we refer to vectors of configuration inputs for an FPGA cell as $CV_i$ and to vectors of operation inputs as $OV_i$. As far as the manufacturer is concerned, test patterns consist of pairs $(TC, TS)$ where $TC$ is a Test Configuration applied at configuration inputs $CV_i$ and $TS$ is a Test Sequence applied at operation inputs $OV_i$. Thus, the manufacturer test procedure is represented as:

$MTP$=$(TC^1, TS^1)...(TC^{nc}, TS^{nc})$ where $TC^i = CV_1^i, CV_2^i.....$

On the contrary, a test pattern for the user consists of pairs $(AC, TS)$ where AC is the application specific Application Configuration, so that the user's test procedure is represented as:

$UTP$=$(AC^1, TS^1)...(AC^{nc}, TS^{nc})$ where $AC^i = CV_1^i, CV_2^i.....$

In the case of $MTP$, $nc$ is the number of possible configurations of an FPGA cell, while in the case of $UTP$, $nc$ is the number of used configurations and it is usually equal to one. Given a non-redundant fault in the unconfigured circuit, it is possible that a specific configuration causes the mentioned fault to become redundant (i.e. no allowable user test sequence can detect it). The definition of a concept of redundancy with respect to a given configuration is therefore needed. Given an FPGA configuration $C$, a fault is considered *C-redundant* (*C-non-redundant*) if it is undetectable (detectable) with respect to the particular configuration $C$.

### 2.2 The Automata used in the test methodology

Given a circuit $c$ and the finite state machine, $M$, that represents it, let $M_f$ be the state machine obtained by injecting a *stuck-at fault* $f$ in $M$. A *test sequence* for fault $f$ is a sequence of input vectors that, when applied to machines $M$ and $M_f$ (both started in their reset states), produces two different output vectors for $M$ and $M_f$. Let us now define $A_f$ as the automaton representing *all* test sequences for fault $f$ in the finite state machine $M$. As an example of the construction of automaton $A_f$, consider the circuit in Figure 1. The fault free state transition graph of state machine $M$ for such

circuit and that of the faulty one, obtained through injection of fault $a$, are shown in Figure 2. The faulty one differs from the fault free by the two dotted transitions. The automaton $A_f$ for fault $a$ is shown in Figure 3. Each node of the graph representing $A_f$ is labeled with a pair of states, $s$ and $s_f$, where the first one is a state of $M$, and the second one a state of $M_f$. An arrow between two nodes indicates that, after application of the input sequence, the fault-free and the faulty machines will be in states $s$ and $s_f$, respectively. A sequence input to $A_f$ is *accepted* whenever the outputs produced my $M$ and $M_f$ in correspondence of the last symbol of the sequence are different. For such detecting sequence the acceptor state is reached in $A_f$. Details on the implementation of the algorithm for constructing the automaton $A_f$ can be found in [2]: we only add here that the operations required, through all steps of the methodology, are carried out implicitly (i.e., through BDD operations, [3]).

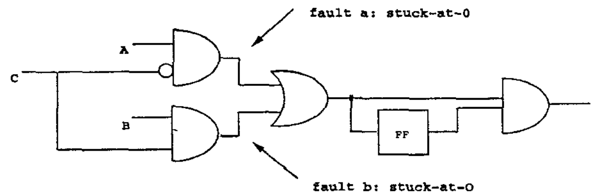


**Figure 1. Sequential circuit used as example.**

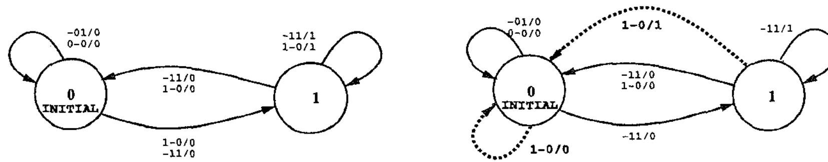


**Figure 2. Fault-Free and Faulty STGs for the Sequential Circuit. Input Vector: (A,B,C).**

We now define the *Global Automaton GAF* [2] as the automaton that accepts the test sequences that individually detect *all* the stuck-at faults of $M$. $GAF$ is therefore obtained by merging all automata $A_{f_1}, \cdots, A_{f_{NF}}$, for all the $NF$ stuck-at faults in M. As an example of construction of automaton $GAF$, let us consider once more our example. For simplicity we show, in Figure 3, the merging of only two automata, $A_{f_1}$ and $A_{f_2}$, obtained by injecting faults $a$ and $b$ shown in Figure 3.

After the construction of the Global Automaton, a phase of compacting the test sequences follows, by the rules introduced in [2]. During this phase, a test sequence of minimal length is obtained through concatenation and overlapping of sequences, one for every stuck-at fault, chosen among *all* test sequences for all stuck-at-faults of machine $M$. If this methodology is applied to an FPGA cell, the obtained automaton $GAF$ recognises all the test sequences for the unconfigured cell, and it

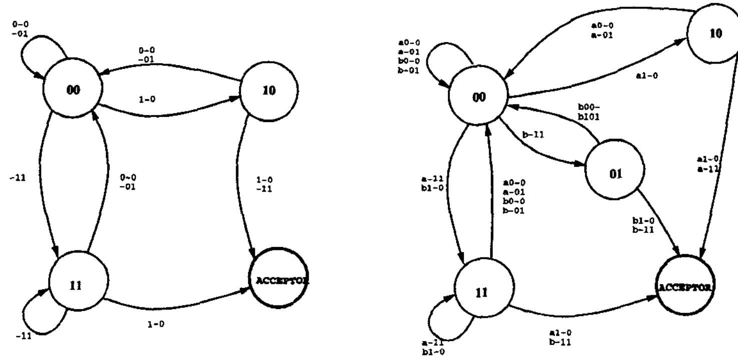


**Figure 3.** $A_f$ **for the circuit, related to fault** $a$**, and** $GAF$ **related to faults** $a$ **and** $b$**.**

therefore represents the *Manufacturer Test Procedure, MTP.* In the next section we show how it is possible, by means of our methodology, to extract the *configuration-specific* test sequences from the test patterns obtained for the unconfigured cell.

## 3 The Proposed Technique: $CGAF$ extraction and manipulation

To be able to extract from $GAF$ the test sequences for the programmed cell, for a given configuration C, and therefore be able to generate configuration-dependent test sequences, we define the *Configured Global Automaton* $CGAF$ as the automaton that represents all the test sequences detecting all *C-non-redundant stuck-at faults* of $M$. To this end, the vectors of configuration inputs, $CV_i$, need to be constrained to value $AC$, while there are no constraints on the value of the vectors of operation inputs, $OV_i$. The *Configured Global Automaton* $CGAF$ is therefore obtained from $GAF$ by deleting all transitions whose values of the $CV_i$s are different from $AC$. Only a subset of the transitions of $GAF$ will be present in the resulting $CGAF$ and therefore a number of states of $GAF$ will be unreachable and a number of faults will possibly be undetectable. Indeed, the set of undetectable faults after extraction of $CGAF$ *is the set of C-redundant faults.* The extraction procedure is done implicitly, by means of Binary Decision Diagrams, thus allowing low complexity in the function manipulation procedures. Indeed, the resulting BDD of $CGAF$ is obtained through an AND operation applied to two diagrams: one representing the characteristic function of the relation $GAF$ while the other representing the boolean function corresponding to configuration C. The complexity of this operation is linear on the product of the number of nodes of the two BDD operands, [3].

Figure 4 shows the $CGAF$ extracted from the previously constructed $GAF$, for the configuration corresponding to C='0'. Note that fault $b$ becomes undetectable (i.e. fault $b$ is C='0'-redundant), while the minimum length sequence for fault $a$ is vector $1-0$ followed by vector $1-0$.

It can be derived from section 2.1 that $CGAF$ represents the $UTP$ for the particular Application

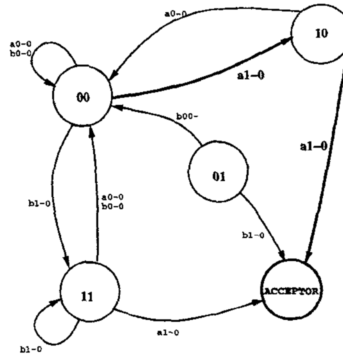


**Figure 4.** $CGAF$ **for Configuration C0 (C='0'). Note that there is no path for fault** $b$ **from the reset (00) to the acceptor:** $b$ **is therefore C0-redundant. On the contrary, a path for fault** $a$ **does exist and is outlined.**

Configuration C, while $GAF$ represents the $MTP$. The novel feature of our methodology resides in the fact that the manufacturer can release the automaton $GAF$, that represents all test sequences for every possible configuration and that *does not show any feature related to the internal structure of the cell*, (see section 4) and the user can extract from it the automaton $CGAF$, for the Application Configuration $C$.

$GAF$ includes *all* sequences for all faults; since, however, for any given fault $f$ to be detected, only one sequence is strictly necessary, we could define a *reduced* $GAF$ that contains one sequence per fault. On the other hand, the more sequences for every fault are included in $GAF$, the better the results of the compacting algorithm will be. The choice of the number of sequences, for every fault, to be included in $GAF$ could therefore be a trade-off between length of the final test sequence and memory available for storage and manipulation of the structure.

The main advantage of this methodology is that the user is given the test sequences for the programmed cell, without needing to know about the structural description of the circuit. The user is indeed able to apply a *structural model* of the fault, without requiring the structural description of the cell. Moreover, since $CGAF$ represents *all* sequences testing the configured cell, the user has a large number of sequences to choose from when accomplishing the task of propagating and justifying the test vectors through the interconnect and among different cells.

The methodology has been applied to an FPGA cell with 38 inputs (8 configuration inputs and 30 operational inputs), 4 outputs and 10 flipflops. The 256 different $CGAF$s are defined as $C_{XX}GAF$ where $XX$ is a hexadecimal number representing the value of the configuration bits. The number of BDD nodes of $GAF$ and of $CGAF$s for 6 different configurations are shown in Table 1. The number of non-redundant faults for the unconfigured circuit and for the 6 configured ones can also be found,

**Table 1. Experimental Results**

| | BDD nodes | (C-)non redundant faults | vectors | generation time |
|---|---|---|---|---|
| $GAF$ | 75059 | 324 | 91 | 2179.8 |
| $C_{00}GAF$ | 23058 | 184 | 44 | 331.96 |
| $C_{01}GAF$ | 17965 | 233 | 55 | 260.57 |
| $C_{0F}GAF$ | 22566 | 265 | 58 | 327.12 |
| $C_{1F}GAF$ | 22213 | 271 | 53 | 274.80 |
| $C_{5F}GAF$ | 23298 | 271 | 55 | 274.80 |
| $C_{FF}GAF$ | 22047 | 213 | 50 | 298.45 |

with the length of the resulting test sequences. Finally, time for $GAF$ and $CGAF$ generation are shown (results were obtained on a Sun Sparcstation 20/125 with 256 Mbyte of RAM and are in seconds). Note that the methodology allows the manufacturer to release one automaton, $GAF$, instead of a relevant number of test sequences (256 in this case).

## 4 Independency of $GAF$ from structural information

Our solution requires the manufacturer to provide users with the complete set of test sequences for the uncommitted (non-programmed) cell of the FPGA. It becomes obviously necessary (in order to grant protection of Intellectual Property rights) to prove that such information will not allow the reconstruction of the cell's specific implementation by reverse engineering techniques. To this end, we first refer to the combinational part of the circuit and to the Boolean Difference Method for identification of all possible test patterns for a given combinational circuit. It will be recalled that, for a single-output function $Z(x_1....x_n)$, injection of a fault $f$ creates a new function $Z_f(x_1....x_n)$ and that solution of equation

$Z(x_1....x_n) \otimes Z_f(x_1....x_n) = 1$

provides all test vectors for $f$.

In the case of multiple output combinational circuit computing functions $Z_1, ..., Z_I$, functions $Z_{i_f}$ are computed after injection of a fault $f$. A test vector for fault $f$ is a configuration of inputs where *at least one* of the $Z_{i_f}$ is complemented with respect to the $Z_i$.

To compute all test vectors for a fault $f$ one should therefore compute the solutions $S_{i_f}$ of the equation system:

$$\begin{cases} S_{1_f} = Z_1 \otimes Z_{1_f} \\ \quad ... \\ S_{I_f} = Z_I \otimes Z_{I_f} \end{cases} \tag{1}$$

and then compute $S_{tot_f}$ from the following:

$S_{1_f} \vee S_{2_f} \vee ... \vee S_{I_f} = S_{tot_f}$

$S_{tot_f}$ represents all test vectors for fault $f$ and GAF indeed recognises the $S_{tot_f}$ for every stuck at fault *class* present in the circuit. Since $S_{tot_f}$ is computed applying the OR operator to multiple operands, it is impossible to uniquely compute such operands $S_{1_f}, S_{2_f}, ..., S_{I_f}$. These represent the only information on the structure of the circuit, a fault $f$ being related to the circuit functionality *and* structure. Not even $S_{i_f}$ being given, the information that can be found in $S_{tot_f}$ is therefore merely functional, and it is not possible to choose one among the different implementations that map to functions $Z_i$. Moreover, note that giving the number of stuck at fault *classes* present in the circuit does not give a unique path from the functionality to the real implementation, since there exist multiple implementations with the same number of fault classes for a given functionality.

The same considerations may be extended to sequential circuits, by using a combinational model. Consider the case of a multiple-output sequential circuit, computing functions $Z_i$ with primary inputs $x_1....x_n$ and flip-flops inputs $s_1....s_m$. By injection of a fault $f$, functions $Z_{i_f}$ are computed. A fault is detected at time $t$ when one of the functions $Z_i(t, x_1(t), ..., x_n(t), s_1(t), ..., s_m(t))$ is complemented with respect to the correspondent $Z_{i_f}(t, x_1(t), ..., x_{n_f}(t), s_{1_f}(t), ..., s_{m_f}(t))$.

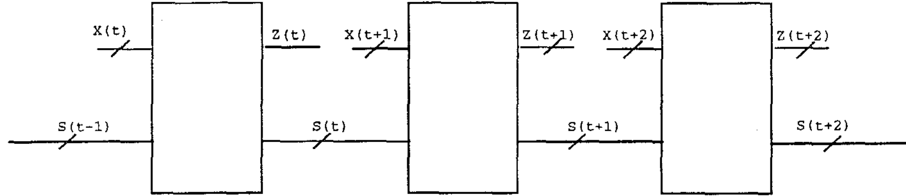


**Figure 5. Combinational model of a sequential circuit**

A combinational model for sequential circuits is constructed by regenerating the feedback signals from previous-time copies of the circuit, as shown in Figure 5. Each time frame is a combinational circuit and an $n$-length sequence of test vectors can be seen as the input of $n$ time frames replica of the corresponding combinational circuit. The presence of an input sequence of length $T$ in $GAF$ means that, at time $T$ :

$Z_i(T, x_1(T), ..., x_n(T), s_1(T), ..., s_m(T)) \otimes Z_{i_f}(T, x_1(T), ..., x_n(T), s_{1_f}(T), ..., s_{m_f}(T)) = 1$

where the values $x_n(T)$ are known to the user (input vector at time T, given in $GAF$) while $s_m(T)$ and $s_{m_f}(T)$ are unknown (internal states of the flipflops, for machine $M$ and faulty machine $M_f$). What has been found for combinational circuits holds (the impossibility to know which and how many outputs are complemented for the presence of the fault); moreover, the values of the state inputs are unknown here.

At time $0 < t < \Gamma$, the user is given the following:

$$Z_i(t, x_1(t), ..., x_n(t), s_1(t), ..., s_m(t)) \otimes Z_{i_f}(t, x_1(t), ..., x_n(t), s_{1_f}(t), ..., s_{m_f}(t)) = 0 \quad \text{for every } i$$

Again, $s_m(t)$ and $s_{m_f}(t)$, the internal states of the flipflops for machine $M$ and faulty machine $M_f$, are unknown. It is therefore not possible to reconstruct the implementation of the circuit when given the sequence detecting a particular class of stuck-at-faults.

## 5 Concluding Remarks

We have presented a methodology for extracting configuration-specific test patterns for FPGA cells, from the set of sequences that test all stuck-at-faults for the unconfigured cell; to the best of our knowledge, no previous investigation has addressed this matter. The user is given the possibility to extract test sequences for the FPGA cell, programmed with the application configuration, without needing to know about the structural description of the circuit. Availability of all significant test sequences for each non C-redundant fault gives the end user a degree of freedom in determining array-level test sequences by suitable choice of inter-cell propagation and justification vectors. Research continues to investigate the propagation of sequences to the primary inputs and outputs of the FPGA, referring to the results obtained in [4] on test generation for interacting finite state machines and in [8] with regards to function mapping while maximising testability of the device. The presented methodology will be applied to an FPGA design that has been developed at the Hewlett-Packard Research Laboratories of Bristol, UK. Alan Marshall and Igor Kostarnov from the mentioned laboratories are acknowledged for the help given.

## References


[1] M. Abramovici, P. Chen, S. Konala, and C. Stroud. Built-in self-test of logic blocks in fpgas (finally, a free lunch: Bist without overhead!). *Proc. IEEE VTS*, 1996.

[2] R. Bevacqua, F. Ferrandi, F. Fummi, and L. Guerrazzi. Implicit test sequences compaction for decreasing test application cost. *Proc. IEEE ICCD*, pages 384–389, 1996.

[3] R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. on Computers*, C-35(8):79–85, August 1986.

[4] F. Ferrandi, F. Fummi, E. Macii, M. Poncino, and D. Sciuto. Test generation for networks of interacting FSMs using symbolic techniques. *Proc. The 6 Great Lakes Symposium on VLSI*, pages 208–213, 1996.

[5] J. Figueras, M. Renovell, and Y. Zorian. Test of ram-based fpga: Methodology and application to the interconnect. *Proc. IEEE VTS*, 1997.

[6] W. K. Huang and F. Lombardi. An approach for testing programmable/configurable field programmable gate arrays. *Proc. IEEE VTS*, 1996.

[7] T. Inoue, H. Fujiwara, H. Michinishi, T. Yokohira, and T. Okamoto. Universal test complexity of field-programmable gate arrays. *Proc. IEEE ATS*, 1995.

[8] I. Pomeranz and S. M. Reddy. Testability considerations in technology mapping. *Proc. IEEE ATS*, pages 151–156, 1994.