JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 16, 733-750 (2000)

Testing Configurable LUT-Based FPGAs

SHYUE-KUNG LU AND JEN-SHENG SHIH
Department of Electronic Engineering
Fu Jen Catholic University
Hsinchuang, Taipei, Taiwan 242, R.O.C.
Email: sklu@ee.fju.edu.tw

A novel approach to testing lookup table (LUT) based field programmable gate arrays (FPGAs) is proposed in this paper. A general structure for the basic configurable logic array blocks (CLBs) is assumed. We group k CLBs in the column into a cell, where k denotes the number of inputs of an LUT. The whole chip is configured as a group of one-dimensional iterative logic arrays of cells. We assume that in each linear cell array, there is at most one faulty cell, and that multiple faulty CLBs in the same cell can be detected. For the LUT, a fault may occur at the memory matrix, decoder, input or output lines. The switch stuck-on and stuck-off fault models are adopted for multiplexers. New conditions for C-testability of programmable/reconfigurable arrays are also derived. Our idea is to configure the cells so as to make each cell function bijective. This property is helpful for applying pseudoexhaustive test patterns to each cell and propagating errors to the observable outputs. In order to detect all the faults defined, $k + 2$ configurations are required, and the resulting number of test patterns is 2^k. A novel built-in self-test structure is also proposed in this paper. The input patterns can be easily generated with a k-bit counter. The number of configurations for our BIST structures is $2k + 4$. Our BIST approaches also have the advantage of requiring fewer hardware resources for test pattern generation and output response analysis. To locate a faulty CLB, three test sessions are required. However, the maximum number of configurations for diagnosing a faulty CLB is $k + 4$.

Keywords: FPGA, fault detection, bijection, built-in self-test (BIST), C-testable

1. INTRODUCTION

Field programmable gate arrays are a popular type of component for emulation and rapid prototyping of complex digital systems. Their applications include microprocessors and telecommunication chips [1]. An FPGA consists of an array of configurable logic blocks (CLBs) along with interconnection channels, and input/output blocks (IOBs). An FPGA can be programmed to implement combinational or sequential logic functions. Because of its low manufacturing cost, short turnaround time and field programmability, the FPGA has been widely used in many applications [1]. Several FPGA architectures have been developed for different applications. The most widely used type is the lookup table (LUT) FPGA, in which the functional unit consists of several lookup tables. This type of FPGA can be reprogrammed any number of times. In this paper, we will focus on the testing of LUT-based FPGAs.

Received June 21, 1999; revised September 10 & December 23, 1999; accepted March 7, 2000.
Communicated by Cheng-Wen Wu.

In recent years, advances in VLSI technology have greatly increased the degree of circuit integration, and rapid development in packaging technology has greatly reduced the controllability and observability of internal nodes. This has significantly complicated testing of systems. If chips are not fully tested after being manufactured, quality may suffer. Therefore, manufacturers of FPGAs all have the same need: an efficient test methodology which will ensure that their products will have high quality. The testing of FPGAs can be divided into two types: testing of unprogrammed FPGAs (configuration-independent testing) and testing of programmed FPGAs (configuration-dependent testing). In configuration-independent testing, no assumptions are made about the function of the FPGA that is configured by the user. On the other hand, configuration-dependent testing involves testing whether a configured FPGA is fault-free. Some techniques for testing FPGAs can be found in [2, 3]. Here, we will focus on the testing of unprogrmmed FPGAs, on which many research results have been published [4-19].

In [4], an array-based technique is proposed to test LUT-based FPGAs. The number of chip configurations used to fully test all CLBs is the same as that needed to test a single CLB with perfect controllability and observability. Universal and C-diagnosable techniques are introduced in [5]. The proposed fault diagnosis procedure is universal–regardless of the logic functions to be realized and the fact that the time required to diagnose an FPGA is independent of the array size. A simple testing arrangement is mentioned in [6]. This approach connects together all the input lines to the CLBs under test from the IOBs, and uses the remaining IOBs to directly observe the outputs. The methodologies proposed in [7, 8] connect each row of an FPGA as a one-dimensional unilateral array for single stuck-at faults. In [9], BIST structures for FPGAs are proposed. The FPGA is repeatedly configured as a group of C-testable iterative logic arrays (ILAs). Test generation and response analysis are performed using internal BIST circuits. In [10, 11, 12], test approaches for interconnects of FPGAs are proposed. Moreover, the testing of multiplexer-based FPGAs can be found in [20].

In this article, we will propose a novel approach to testing LUT-based FPGAs. A generalized structure for the basic configurable logic blocks is assumed. This structure can be found in various Xilinx FPGA families. Since an FPGA is structured as an $N \times N$ array of CLBs, it can be naturally viewed as a two-dimensional array or as N one-dimensional arrays. Unfortunately, a CLB does not have enough outputs to drive the inputs of a neighboring CLB. This makes it difficult to configure an FPGA as an ILA. Therefore, if we treat an FPGA as an $N \times N$ array of CLBs directly, then we cannot exploit the testing advantages of ILA's. Fortunately, this problem have been solved in [9], where some CLBs are used as helpers−auxiliary cells whose goals are to generate "locally" the missing inputs and to provide separate propagation paths for the output responses of blocks under tested (BUTs). However, although some portion of the helper CLB may also be tested alongside the BUT, it must be fully tested in a different test session. Therefore, more chip reconfigurations and test patterns are required to complete the test process.

In order to deal with this problem, a novel cell structure is proposed here. In this paper, a column of k CLBs is grouped into a cell, where k denotes the number of inputs of an LUT. Then, the whole chip is partitioned into N/k disjoint one-dimensional arrays of cells. We assume that in each linear cell array, there is at most one faulty cell, and that a faulty cell may contain multiple faulty CLBs. In other words, multiple faulty cells can be detected if they exist in different linear arrays. By using this cell structure, the test pattern generator and the output response analyzer can be easily implemented. All the cells in the FPGA can be tested simultaneously to speed up the test process. Moreover, since each cell is locally connected to its neighboring cells, there requires no considerable demands on the global routing resources. This increases the probability of successful routing [9]. During testing, the configurations of each cell will make its function bijective, which is helpful for applying pseudoexhaustive test patterns to each cell and propagating errors to observable outputs. In order to detect all the faults modeled, $k + 2$ configurations are required. For each configuration, a minimal complete input sequence (consisting of all input combinations for a cell) is applied to the leftmost cells of each linear array from IOBs, and the outputs of the rightmost cells can be observed directly from the remaining IOBs. A novel built-in self-test structure is also proposed in this paper. The input patterns can be easily generated with a k-bit counter and the output response analyzed with a checksum analyzer or comparator. Our BIST approaches also have the advantage of requiring fewer hardware resources for test pattern generation and output response analysis. The number of configurations for our BIST structures is $2k + 4$. To locate a faulty CLB, three test sessions are required. However, the maximum number of configurations for diagnosing a faulty CLB is $k + 4$.

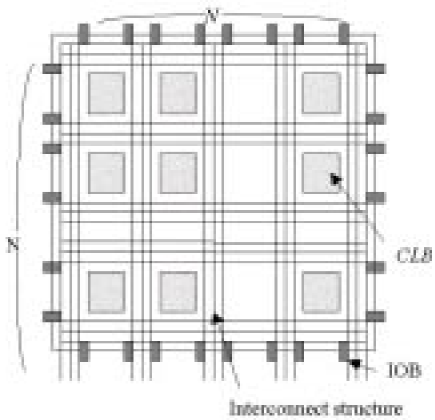

Fig. 1. FPGA architecture.

2. PRELIMINARIES

Fig. 1 shows the FPGA architecture considered in this paper. The FPGA can be considered as an $N \times N$ array of CLBs. The CLBs can be programmed to implement combinational and sequential logic functions. All the CLBs are identical before they are configured.

There are also programmable interconnection networks and input/output blocks to provide an interface between the package pins and the internal circuits. The structure of a CLB is shown in Fig. 2, which consists of one lookup table, two multiplexers (MUX1 and MUX2), and one D flip-flop (DFF). Let k denote the number of input lines of an LUT; then an LUT can implement 2^n logic functions, $n = 2^k$. The configuration memory cells (CMCs) are used to configure its logic functions.

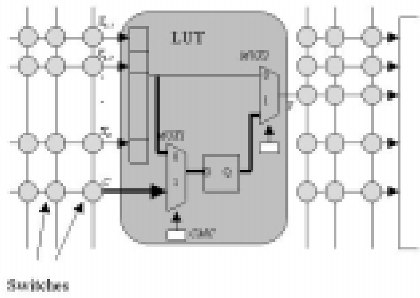

Fig. 2. The structure of a CLB.

When programming an FPGA, we load the bit patterns corresponding to the function's truth table into the configuration memory cells and also configure the interconnection networks. Such a programming process is called a configuration. It is well known that an unprogrammed FPGA can be configured to implement different logic functions. Therefore, testing this category of FPGAs implies that testing of all possible functions can be realized. The procedure for testing CLBs involves repeatedly implementing a test configuration (TC) and alternately applying test patterns (TPs) to this configuration. Since an LUT can realize 2^n different functions, it is impractical to test each function exhaustively.

Fortunately, with our approach, the number of test patterns required for each test configuration is the same–a minimal complete input sequence (defined later) of a cell. The cell array is pseudoexhaustively tested by applying exhaustive tests to each cell. Our test patterns are independent of the size of the FPGA, i.e., they scale well with increasing FPGA size. Therefore, the remaining problem is to determine how to configure the cells in order to detect all the faults modeled using the defined fault models. It is evident that the task of test configuration is very time-consuming. In general, the time needed for test configuration is much greater than that needed to apply test patterns. Therefore, to test an FPGA, we must seek out an efficient approach which requires fewer TCs and covers all the programmable resources of FPGAs. In other words, the proposed approach must speed up the test process and guarantee the quality of the products being tested. It should also be noted that the proposed approach must not place considerable demands on the global routing resources within the FPGA, and that it should increase the probability of successful routing.

In general, the number of inputs, k, of a CLB is larger than that of its output counts. As described in the previous sections, it is not beneficial to view an FPGA as an $N \times N$ array of CLBs. This is because a CLB does not have enough outputs to drive the inputs of a neighboring CLB, so it is impossible to construct an iterative logic array directly. If we treat an FPGA as an $N \times N$ array of CLBs, we cannot exploit the testing advantages of ILA's. Therefore, many test approaches designed for ILA's cannot be applied in FPGA testing. To solve this problem, we define a cell as a column of k neighboring CLBs with their corresponding input lines tied together. In other words, a cell is a $k \times 1$ array of CLBs. This cell structure is shown in Fig. 3, where each CLB contains k input lines, $x_0, x_1, ..., x_{k-1}$, and one output line, y_i, $0 \le i \le k-1$. Therefore, we can regroup an FPGA consisting of an $N \times N$ array of CLBs into an $N/k \times N$ array of cells. An FPGA consists of a CLB array as shown in Fig. 4, where CLB_{ij} denotes the CLB in row i and column j, $0 \le i, j \le N-1$. The cell array corresponding to that in Fig. 4 is shown in Fig. 5. The outputs of one cell can be used to

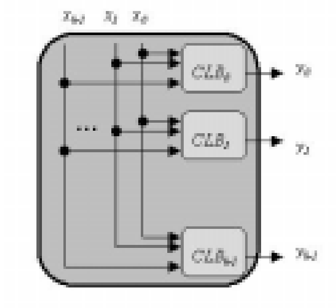

Fig. 3. Architecture of a cell.

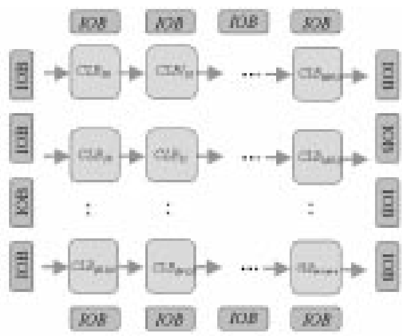

Fig. 4. An $N \times N$ FPGA array.

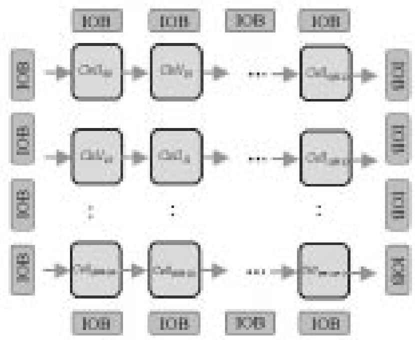

Fig. 5. The cell array corresponding to that in Fig. 4.

drive its neighboring cells. Therefore, N/k disjoint linear arrays of cells are obtained, and their inter-cell communications are local (between adjacent cells). This will not violate the routing limitations of FPGAs, and it will increase the probability of successful routing. This is different from the approach proposed in [9], where helpers are used to generate "locally" the missing inputs and to provide separate propagation paths for BUT output responses. These outputs cannot be used as input patterns for the next BUT in the iterative logic array. One of the main advantages of our approach is that all the cells in the same test session are tested simultaneously. However, the helpers used in [9] must be tested in a different session. Therefore, our approach will result in fewer test configurations and will significantly speed up testing.

We assume that at most one faulty cell exists in each linear cell array. Otherwise, fault masking might occur. However, multiple faulty cells that exist in different linear arrays can also be tested since all linear arrays are tested independently. For a faulty cell, the faulty cell function can deviate in any manner, and the cell can contain multiple faulty CLBs. Moreover, we assume that the interconnection networks are tested first. Related approaches to the testing of interconnects can be found in [12-15]. The fault model adopted in this paper is described physically as follows, and it has been shown to be suitable for FPGAs [5, 6].

- For a lookup table, a fault may occur at the memory matrix, decoder, inputs and outputs of a CLB. A faulty memory matrix has some memory cells that are incapable of storing the correct logic values (i.e., stuck-at 1 or stuck-at 0 may occur at a memory cell). If a fault occurs at the decoder, then incorrect access, non-access and multiple access faults may occur. For the input and output lines, stuck-at faults are considered.

- A multiplexer is a group of switches controlled by the configuration memory bits. Only one switch is allowed to be on. Either none or more than one on-switch is invalid. If all the switches are off, we assume the output is either stuck-at-1 or stuck-at-0. Alternatively, if two switches are on simultaneously, then the output is the logic wired-OR of the selected inputs. In other words, switch stuck-on and switch stuck-off fault models [18] are adopted. For a D flip-flop, we adopt the functional fault model. A fault may cause the flip-flop to be unable to receive data, to be triggered by the correct clock edge, or to be incapable of being set or reset [8].

3. TEST STRATEGIES FOR FPGAS

In this section, we will present our proposed test strategies for FPGAs. New conditions for C-testability of programmable/configurable arrays will be proposed. With our approach, all the cells in an FPGA can be tested concurrently as described in the following. In order to simplify our discussion, some definitions will be given first.

Definition: An unprogrammed cell is a $k \times 1$ array of CLBs. Each CLB contains a lookup table which can implement 2^n functions, $n = 2^k$.

Definition: A programmed cell with the function f is a combinational machine (Σ, Δ, f), where f: $\Sigma \rightarrow \Delta$ is the programmed cell function and $\Sigma = \Delta = \{0, 1\}^k$, where k denotes the number of inputs of a CLB.

Definition: A complete or exhaustive input sequence σ for a programmed cell with the function f is an input sequence consisting of all possible input combinations for the cell. A complete output sequence is defined analogously.

Definition: A minimal complete input sequence θ for a cell is the shortest complete input sequence (which has a length of 2^k, i.e., $\theta = \delta_1\delta_2 \ldots\delta_n$).

Definition: We say that the function f of a programmed cell is bijective when $\forall \delta_1 \neq \delta_2$, $f(\delta_1) \neq f(\delta_2)$, and that the length of the minimal complete input sequence and that of the minimal complete output sequence are identical, i.e., $|\Sigma| = |\Delta|$.

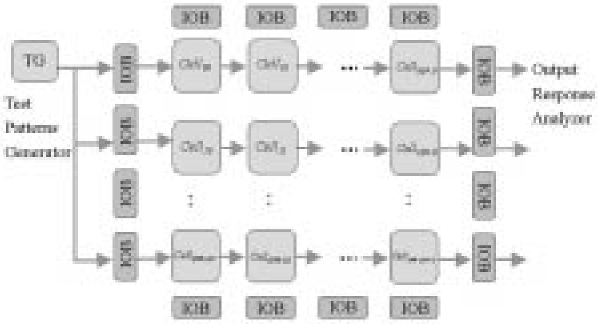

Fig. 6. The connection structure.

A. Testing of LUT's

We consider initially that a CLB consists of a single LUT. Then, all the input lines of the leftmost cells and the output lines of the rightmost cells for each linear array are connected to IOBs, so that they can receive their test patterns and observe the corresponding outputs, respectively. The number of IOBs required for inputting test patterns and observing the results is at most $2N$. If all the linear arrays use the same IOBs to input test patterns, then the number of IOBs will be reduced to $N + k$. This number will not tend to exhaust the limited I/O resources available at the periphery of the chip. During testing, the k output lines of $cell_{pq}$ are appropriately connected to the k input lines of $cell_{p(q+1)}$, $0 \leq p \leq N/k - 1$ and $0 \leq q \leq N - 1$. This connection structure is shown in Fig. 6, where each array receives its test patterns from the TG (test pattern generator) through the same IOBs. The outputs are then sent to the output response analyzer so that the responses can be analyzed in order to produce a pass/fail indication.

After the interconnection network has been configured, there are still three remaining questions to be answered before the LUT's can be tested. (1) Which test set should we apply to test all the linear arrays and ensure acceptable fault coverage? (2) Verifying a cell function involves generating inputs for each cell (i.e., controlling the cell) and propagating faults from the cell (i.e., observing the cell). How can we solve the controllability and observability problems for FPGAs? (3) Unlike general iterative logic array structures, each cell in an FPGA can be configured to have up to 2^n functions. What functions should we choose to ensure their fault detection capability and minimize the number of configurations. The first question can be answered by applying pseudoexhaustive patterns to each cell. The answers to the last two questions can be found in the following paragraphs.

Besides control signals, each cell has *k* input lines and output lines, respectively. The function of a cell can be described with a truth table. If the cell function satisfies the bijectivity conditions, then testing each linear array is easy. Therefore, we can configure a cell so as to make its function bijective. The LUT input is assumed to be represented by the k-bit word, $x_{k-1}\,x_{k-2}\ldots x_0$, while the cell output is represented by the word $y_{k-1}\,y_{k-2}\ldots y_0$. If β is a feasible configuration of values for the output entries of the truth table, then β_i denotes β confined to the output variable y_i, $0 \le i \le k-1$. In other words, β_i is configured with only the values for y_i and $\beta = \beta_{k-1}\,\beta_{k-2}\ldots\beta_0$. We also assume that the truth table is fully expanded, i.e., there are no don't-care terms. Next, we produce a feasible test configuration, β^0, for each cell, which contains all the possible binary k-bit values. In other words, the cell function is configured to be bijective. Moreover, the test configuration β^k is the complement of configuration β^0. An example of a bijective configuration with $k = 3$ is shown in Fig. 7. In this example, the configuration is for the variables $y_2 = 00001111$, $y_1 = 00110011$, and $y_0 = 01010101$, respectively.

Without loss of generality, we can re-permute the output variable assignments so as to obtain other configurations required to completely test a cell. In other words, the output assignments for configuration β^i can be expressed as

$$\beta^i = \beta_{(k-1+i) \bmod k}\,\beta_{(k-2+i) \bmod k}\ldots\beta_{(i) \bmod k},\ 1 \le i \le k-1.$$

We will use the assignment shown in Fig. 7 as an example, where $\beta^0 = \beta_2\beta_1\beta_0$, $\beta^1 = \beta_0\beta_2\,\beta_1$, and $\beta^2 = \beta_1\beta_0\beta_2$.

X_2	X_1	X_0	Y_2	Y_1	Y_0
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	0
1	1	1	1	1	1

Fig. 7. An example configuration whose outputs contain all 8 possible binary 3-bit values.

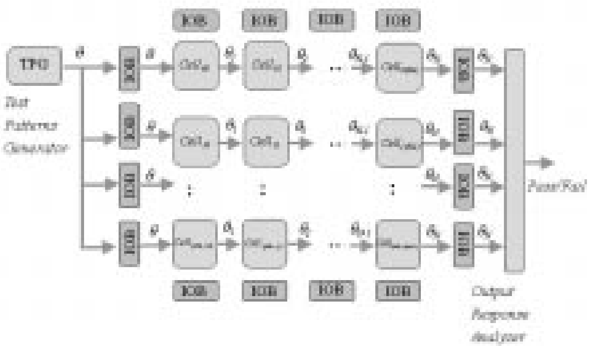

Fig. 8. Application of test patterns.

Observation: The output configuration contains all the possible binary k-bit values. That is, the cell function with configuration α^i is also bijective.

Proof: This follows directly from the definition of a bijective cell function. □

Observation: For a fault-free cell, if a minimal complete input sequence θ is applied to the inputs of a cell with configuration β^i, $0 \le i \le k$, then the output sequence will also be a complete sequence if it is fault-free.

Proof: Since the cell function is also bijective with the configuration β^i, $0 \le i \le k$, it follows that if the cell is fault-free, the outputs will also be a complete sequence from the definition of a bijective function. □

Theorem 1: If a minimal complete input sequence θ is applied to the inputs of a cell with configurations β^k and β^0, respectively, then all the stuck-at faults in the LUT memory cells can be detected with 100% fault coverage. □

Proof: Assume that a stuck-at-0 (1) fault occurs at the i'th memory cell in CLB_j of a cell; then, the i'th entry of v_j will have the same value for configurations β^0 and β^k. However, since β^k andβ^0 make the cell function bijective and their corresponding assignments are complemented, this fault will make either β^0 or β^k to not be bijective. It is evident that this fault can be detected by examining whether the output is a complete output sequence. Therefore, we conclude that all stuck-at faults in the LUT can be detected with 100% fault coverage.

Theorem 2: If a minimal complete input sequence θ is applied to the inputs of a cell with configurations β^{k-1}, β^{k-2}, ..., and β^0,respectively, then any fault within the decoder can be detected.

Proof: If a fault occurs within the decoder, then incorrect access, non-access or multiple access faults may occur. All these faults must be detected by θ. The proofs for each type of faults are given in the following.

1. Incorrect access faults

If the decoder cannot generate the correct memory address, then a faulty memory address is accessed. That is, if we try to access the i'th memory cell of a LUT, then the j'th memory cell will be accessed if there exists a incorrect access fault, $i \ne j$, $0 \le j \le 2^k - 1$. A large number of such faults may exist in a cell. To test this type of fault, we must first activate this fault. In other words, we must find a configuration such that the values stored in the i'th and j'th memory cells (MCs) are different, i.e., $MC(i) \ne MC(j)$. Fortunately, with the configurations β^{k-1}, β^{k-2}, ..., and β^0, there exists at least one such configuration that the accessed values of the correct and faulty memory addresses will have different stored values. If the fault is activated by a specific configuration, then its corresponding outputs will not again be a complete output sequence. This is useful to detect this fault. Therefore, we conclude that all the incorrect access faults can be detected with 100% fault coverage.

2. Non-access faults

If a non-access fault exists, then the output of the LUT will stick at 0 (1). Then, we can treat it as a stuck-at 0(1) fault. This type of fault can be detected with configurations β^k and β^0 as described in the previous paragraph. Therefore, we conclude that non-access faults can be detected with 100% fault coverage.

3. Multiple access faults

If a multiple access fault occurs within a CLB, then the output value of this CLB is equal to the wired-OR (wired-AND) of the stored values of all the accessed addresses. That is, the output value $y = MC(i) + MC(j)$ (wired-AND) or $y = MC(i) \bullet MC(j)$, where $MC(i)$ and $MC(j)$ denote the values of the correct and extra accessed memory cells, $i \neq j$, $0 \leq j \leq 2^k - 1$. To activate this type of fault, there must exist a configuration such that $MC(i) = 0$ and $MC(j) = 1$ (wired-OR) or $MC(i) = 1$ and $MC(j) = 0$ (wired-AND). Then the wired-AND (wired-OR) output of this CLB will have logic value 0/1 (wired-AND/wired-OR) and the faulty effect is activated. Fortunately, the configurations β^{k-1}, β^{k-2}, ..., and β^0 can be used to activate this type of fault. The outputs of the faulty cell will no longer be a complete sequence and can be observed from the cell's outputs. We conclude that all the multiple access faults can be detected with 100% fault coverage. □

B. Testing of D flip-flops

D flip-flops are tested using the functional model. That is, D flip-flops are treated as pipelined latches. A fault may cause a flip-flop to be unable to receive data, to be triggered by the correct clock edge, or to be incapable of being set or reset. With our configurations β^0 through β^k, there exists at least one configuration which contains the sequences 010 and 101. It should be noted that the configuration memory must be configured so as, to direct the outputs from the LUT through the highlighted path to the CLBs output. Fortunately, this will not increase the number of configurations and can be done during the process of testing the LUT's. Therefore, D flip-flops can be used as pipelined latches and can be fully tested using the same configurations and test patterns that are used for an LUT.

C. Testing of multiplexers

As with the fault model defined above, switch stuck-on and stuck-off faults are assumed. For stuck-off faults, we assume that the output is either stuck-at 1 or stuck-at 0, and that it can be collapsed like stuck-at faults at the output line of an LUT. This type of fault can be simultaneously tested as in the testing of an LUT according to Theorem 1. For stuck-on faults, wired-operations are performed, and the faults can be detected in the same manner as described in Theorem 2.

Now that we have proven the testability conditions for a single cell, we can turn to the problem of testing a whole array. The first task is to send the test patterns to each cell in an FPGA (controllability) and propagate the fault effects to the primary outputs (observability). The controllability and observability problems can be solved using the following theorem.

Theorem 3: For an FPGA cell array, all the faults defined can be tested with a minimal complete input sequence and configurations β^k, β^{k-1}, ..., and β^0 for each cell.

Proof: As shown in Theorem 1 and Theorem 2, a complete input sequence and test configurationsβ^i, $0 \le i \le k$, can be used to test all the faults within a cell. For an entire cell array, we must solve the controllability and observability problems. First, all the cells must have the same configurations during each test phase; i.e., they are configured to have the same bijective cell function. Since all the linear arrays are identical, we connect all the inputs of the left-most cells to IOBs and use an external test pattern generator (TG) to generate test patterns for all the arrays as shown in Fig. 8. Since all the input IOBs also receive a minimal complete input sequence, the faults that occur in an IOB can also be detected. The output IOBs can also be tested in the same manner. The input to each linear array is a minimal complete sequence, so the simplest approach to implementing the test pattern generator is to use a k-bit binary counter, which counts from 0 to $2^k - 1$.

It is well known that, a minimal complete input sequence is sufficient to completely test a cell. Let θ be a minimal complete input sequence. We apply θ to the input of the leftmost cell of each linear array. Since each cell is configured as a bijective cell function, it follows that if $cell_{00}$ is fault-free, the output sequence θ_1 is also a minimal complete sequence. This sequence can then be used as the input sequence for $cell_{01}$, obtaining complete sequence θ_2. We can then apply θ_2 to $cell_{02}$, producing complete sequence θ_3. By reiterating this process, we can construct a complete input sequence for each cell in the linear array. The same mechanism can be applied to the other linear arrays. This solves the controllability problem.

Now, we will turn our attention to the observabiltiy problem. If $cell_{00}$ is faulty, a faulty effect must occur at the outputs of this cell. Then, since the cell function is bijective, the faulty effect will be propagated through $cell_{01}$ and appears at its outputs. This is because any input change is propagated to an output change. Bijectivity ensures that the change will continue to ripple to the outputs of the rightmost cell, which in turn are connected to the IOBs and output response analyzer for observation. The output sequences of all the linear arrays are identical if no faults exist in the FPGA chip. The output response analyzer is simply a comparator and can be used to compare these outputs and produce a pass/fail indication. Other types of analyzers can also be used. For example, a checksum checker can be used to check the sum of the output sequence. Since we assume that there exists at most one faulty cell in a linear array, the propagating path cannot mask each fault. We conclude that the cell faults in each linear array can be detected using the complete input sequence. □

An additional configuration must be added to test the c input of MUX1. With this configuration, the output of the LUT is forwarded to the input of MUX1. The c input is propagated through the highlighted path as shown in Fig. 2 to the output y and used as input for the neighboring cell. Therefore, the number of configurations needed to completely test the whole FPGA array is $k + 2$.

4. Built-In Self-Test

The proposed BIST structure is shown in Fig. 9, where some of the cells are configured as test pattern generators (TPGs) and some as output response analyzers (ORAs). The remaining cells are cells under tested (CUTs). The TPG is a simple k-bit binary counter, which provides pseudoexhaustive test patterns to the leftmost CUT of each linear array. As

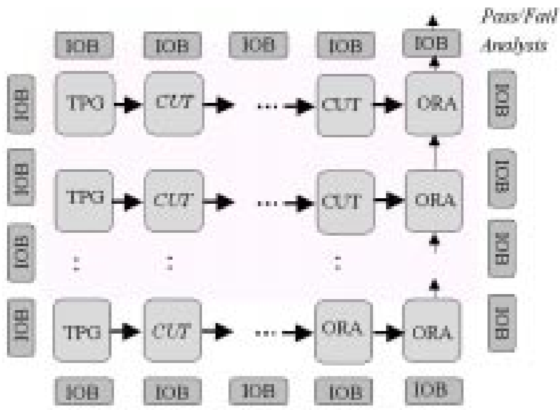

Fig. 9. Built-in self-test structure of FPGAs (1st session).

a result of the tessellation of test patterns described above, each CUT will receive exhaustive test patterns. The outputs of the rightmost CUTs are the same if all the CUTs are fault free. Due to the special characteristic of the test patterns–a minimal complete input sequence for each cell–there are novel techniques for output response analysis. The first one is the check-sum method: each ORA in Fig. 9 is configured as an accumulator, i.e., a binary adder. Since the output response of the rightmost CUT is also a minimal complete sequence, the accumulated output becomes

$$\sum_{i=1}^{n} \delta_i = \frac{2^k(2^k-1)}{2} = 2^{k-1}(2^k-1) = 2^{2k-1} - 2^{k-1}.$$

For example, if $k = 4$, then the accumulated result is 111000. Judging from the bit positions of zeros and ones, one can determine immediately whether the output sequence is faulty or not. This approach can detect all the single-bit errors, resulting in higher fault coverage. The only undetected errors are those multi-CLB faults in different CLB rows that generate different permutations of the correct output sequence.

Another approach is to use parity checkers. A minimal complete sequence implies that the number of 1s and 0s in the sequence, at each bit position, must be equal. That is, even parity must be true for every output bit. Fig. 10 shows the logic diagram of a simple parity checker. The short thick segment stands for a unit delay, which can be implemented using the internal flip-flop of a CLB. This approach requires fewer FPGA resources than does the previous one and can achieve relatively high coverage. The third approach is to use equality checkers. Based on the tessellation property of the test patterns, the output sequences from the rightmost CUTs are identical. We can compare these sequences using an equality checker, which is composed of XOR gates. In each of the analysis approach, the

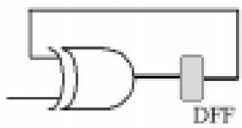

Fig. 10. The parity checker.

outputs of the ORAs can be sent to a shift register (SR) chain, which performs parallel-to-serial conversion. The results are then shifted out sequentially for external go/no-go judgement. These BIST structures nearly exclude the need for an external ATE (automatic test equipment).

Our BIST structure contains two test sessions. The first session is shown in Fig. 9. In the first test session, the CUTs are tested with various test configurations as described in section 3. Then, we have to swap the functions of TPGs and ORAs with those of CUTs such that each CUT will be tested in at least one test session. This goal is achieved by using the second test session as shown in Fig. 11. The test configurations are the same as that for session 1. In this session, we assure that each CUT is tested at least once. The number of configurations for our BIST structures is $2k + 4$ since two test sessions are required. This is better than [9], where 3 test sessions are required.

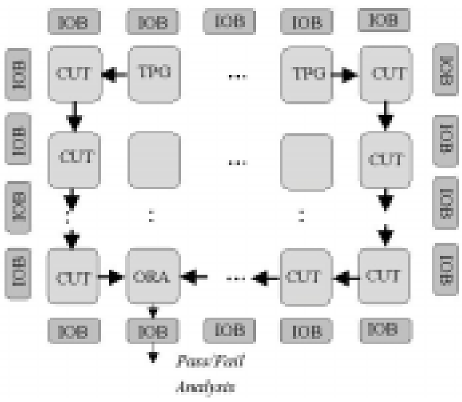

Fig. 11. 2nd test session.

5. FAULT DIAGNOSIS

FPGA fault diagnosis is also an important task. If we can locate a faulty CLB, then the fault-tolerant mechanism can be activated to bypass it, and the FPGA can implement its logic function using only fault-free CLBs. Our FPGA diagnosis process consists of a sequence of steps: (1) horizontal diagnosis, (2) vertical diagnosis, and (3) CLB diagnosis. Step 1 and step 2 can be used to locate faulty cells. Since our diagnosis resolution is one CLB, step 3 must be added for CLB diagnosis.

- Step 1 (horizontal diagnosis): For horizontal diagnosis, the corresponding interconnect structure is shown in Fig. 12(a). The outputs of the rightmost cells are compared with the correct results to identify faulty rows.
- Step 2 (vertical diagnosis): The interconnect structure is changed to that shown in Fig. 12 (b). The outputs of the bottommost cells are compared with correct results for identifying faulty columns. Note that multiple cells can be detected if they are in different rows (columns).

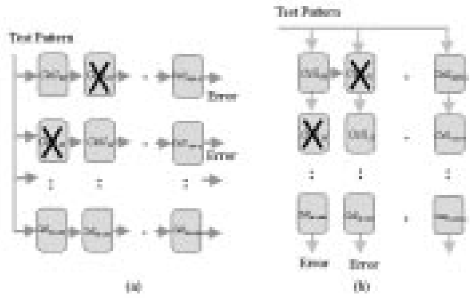

Fig. 12. (a) Horizontal diagnosis and (b) vertical diagnosis.

- Step 3: After step1 and step 2 are performed, faulty cells can be located by means of intersecting faulty rows and faulty columns. In order to locate faulty CLBs, we reconfigure the cells right to the faulty cells and bypass its logic as shown in Fig. 13. This provides us direct observation of faulty cells. The other cells retain their original configuration. Due to the special characteristics of the test patterns, the output sequence of the rightmost cells can be predicted before test patterns are applied if faulty cells do not exist. Then, we can compare the output sequence at each bit position with the correct sequence to locate faulty CLBs. For example, if y_i is different from the correct sequence, then the CLB_i of the faulty cell is faulty.

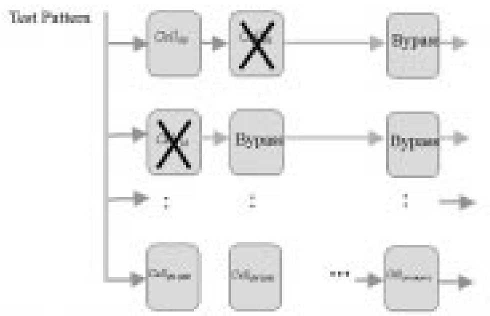

Fig. 13. CLB diagnosis.

6. EXPERIMENTAL RESULTS

In order to establish, verify, and evaluate our testing approach, a fault simulator was implemented using the *C* language for the generic FPGA architecture. This architecture can be easily applied to the FPGA families from Xilinx [21], e.g., XC3000, XC4000, and XC5200, with minor modifications. Note that the fault simulation information, including the number of configurations and the fault coverage, is for a single cell. The experimental results for k = 4 are summarized in Table 1. From this table, we find that the total number of faults within a cell is 1696, and that all these faults can be detected with 6 configurations. The

Table 1. Test configurations and test patterns for *k* = 4.

	Lookup Tables				Multiplexers				Stuck-at Faults	DFFS	Total
	Functional Faults	Multiple Access Faults	Incorrect Access Faults	Non-Access Faults	Multiple Stuck-on Faults	Incorrect Switch-on Faults	Stuck-off Faults	Functional Faults			
Number of Faults	32	480	960	8	8	16	16	16	144	16	1696
Number of Test patterns	16	16	16	16	32	32	2	32	32	32	32
Number of Faults detected	32	480	960	8	8	16	16	16	144	16	1696
Number of Configurations	4	4	4	1	2	3	2	2	6	2	6
Faults Coverage	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%

required number of test patterns is 16 (a complete input sequence for a cell) for almost all test configurations. It should be noted that not all types of faults require the use of 6 configurations to complete the test. For example, the number of configurations needed to completely detect the functional faults of an LUT is 4, and 2 configurations are required to test DFFs. The row “Number of Configurations” has different values for each fault type. However, the total number of test configurations is only 6 since they can be repeatedly used for different fault types. For multiplexers, more fault types than were defined previously were simulated. These include multiple stuck-on faults, incorrect switch-on faults, stuck-off faults, and functional faults.

The relationship between the number of configurations and the fault coverage is shown in Fig. 14. For example, if only 1 configuration is programmed, the resulting fault coverage for functional faults is only 48%. However, if 4 configurations are programmed, then all 32 functional faults are detected, and the accumulated fault coverage is 100%. We compared our approach with that in [5] since the same CLB structure is used. However, instead of the single-CLB fault model used in [5], multiple-CLB faults can be detected with

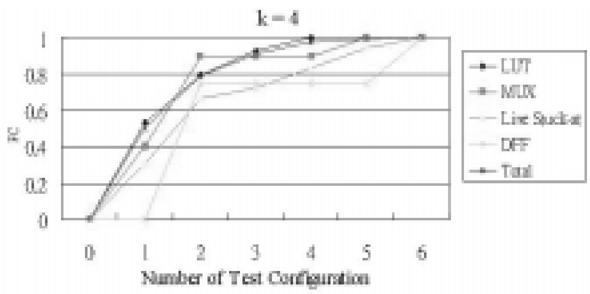

Fig. 14. The relationship between the number of configurations and the fault coverage.

Table 2. Comparisons

APPROACH	GOAL	NTP	NTC	NCD	NTS	REGULARITY
T. Inoue [5]	Fault Diagnosis	2^k	$2k + 4$	$4k + 8$	–	Low
C. Stroud [9]	Built-In Self-Test	2^k	–	–	3	Low
This paper	Fault Detection, BIST, and Diagnosis	2^k	$k + 2$	$k + 4$	2	High

our approach. Table 2 summarizes the results obtained in this study. In this table, we list the goals of various studies, and compare the number of test patterns required for each configuration (NTP), the number of test configurations required for fault detection (NTC), the number of configurations for fault diagnosis (NCD), and the number of test sessions for BIST (NTS). The number of test configurations for fault detection (NTC) is only half that required in [5]. The parameter "Regularity" denotes the complexity of the corresponding BIST structures. From this table, we can find that our approach improves the detection time significantly. Moreover, the number of test sessions is smaller than that in [9] if the BIST structure is considered.

7. CONCLUSIONS

A novel approach to testing field programmable gate arrays based on lookup tables has been proposed in this paper. A column of k CLBs is grouped into a cell, where k denotes the number of inputs of an LUT. We use a cell as the basic test element instead of using a CLB. The whole chip is partitioned into disjoint one-dimensional arrays of cells. We assume that in each linear array, there is at most one faulty cell, and that a faulty cell might contain multiple faulty CLBs.

For a faulty lookup table, a fault may occur at the memory matrix, decoder, input and output lines. Stuck-on and stuck-off fault models are adopted for multiplexers. Moreover, we assume that the interconnection network has been tested. Our idea is to configure each cell function as bijective. We need $k + 2$ configurations to test all the faults defined. The number of configurations is less than that in previous works, and the test time can be significantly reduced. For each configuration, a minimal complete input sequence is applied to the leftmost cells of each linear array from the IOBs, and the outputs of the rightmost cells can be observed directly from the remaining IOBs by using an output response analyzer. The complete input sequence can be easily generated using a k-bit binary counter. Experimental results show that the resulting fault coverage is 100%, and that the time required to test the whole chip is independent of the array size. Our approach is very suitable for BIST structures since the test pattern generator and the output response analyzer are easier to implement. The routing complexity and the required number of IOBs are also reduced significantly. To locate a faulty CLB, three test sessions are required. However, the maximum number of configurations for diagnosing a faulty CLB is $k + 4$.

REFERENCES

1. S. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic, *Field Programmable Gate Arrays*, Kluwer Academic Publishers, Boston, MA, 1992.
2. W. Quddus, A. Jas, and N. A. Touba, "Configuration self-test in FPGA-based reconfigurable systems," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, 1999, pp. I-97-I-100.
3. W. Quddus, "Configuration self-test in FPGA-based reconfigurable systems," Master Thesis, Dept. of ECE, University of Texas at Austin, May 1999.
4. W. K. Huang, F. J. Meyer, X. T. Chen, and F. Lombardi, "Testing configurable LUT-based FPGAs," *IEEE Transactions of VLSI Systems*, Vol. 6, No. 2, 1998, pp. 276-283.
5. T. Inoue, S. Miyazaki, and H. Fujiwara, "Universal fault diagnosis for lookup table FPGAs," *IEEE Design & Test of Computers*, 1998, pp. 39-44.
6. W. K. Huang, F. J. Meyer, X. T. Chen, and F. Lombardi, "A general technique for testing FPGAs," in *Proceedings of IEEE VLSI Test Symposium*, 1996, pp. 450-455.
7. W. K. Huang, F. J. Meyer, and F. Lombardi, "Array based testing of FPGAs: Architecture and complexity, general technique for testing FPGAs," in *Proceedings of IEEE Conference on Innovative System Silicon*, pp. 249-258.
8. T. Liu, W. K. Huang, and F. Lombardi, "Testing of uncustomized segmented channel FPGAs," in *Proceedings of ACM Symposium on FPGAs*, 1995, pp. 125-131.
9. C. Stroud, P. Chen, S. Konala, and M. Abramovici, "Using ILA testing for BIST in FPGAs," in *Proceedings of IEEE VLSI Test Symposium*, 1995, pp. 256-261.
10. T. Inoue et al., "Universal test complexity of field programmable gate arrays," in *Proceedings of Fourth IEEE Asian Test Symposium*, 1995, pp. 259-265.
11. M. Renovell, J. Figueras, and Y. Zorian, "Test of RAM-based FPGAs: Methodology and application to the interconnect," in *Proceedings of 15th IEEE VLSI Test Symposium*, 1997, pp. 230-237.
12. M. Renovell, J. M. Portal, J. Figueras, and Y. Zprian, "Testing the interconnect of RAM-based FPGAs," *IEEE Design & Test of Computers*, 1998, pp. 45-50.
13. S. J. Wang and C. N. Huang, "Testing and diagnosis of interconnect structures in FPGAs," in *Proceedings of 7th Asian Test Symposium (ATS '98)*, 1998, pp. 283-287.
14. S. J. Wang and T. M. Tsai, "Test and diagnosis of fully logic blocks in FPGAs," *IEE Proceedings - Computers and Digital Techniques*, Vol. 146, No. 2, 1999, pp. 100-106.
15. F. Lombardi, D. Ashen, X. T. Chen, and W. K. Huang, "Diagnosing programmable interconnect systems for FPGAs," in *Proceedings of International Symposium on FPGAs*, 1996, pp. 100-106.
16. C. Stroud, S. Konala, P. Chen, and M. Abramovici, "Built-in self-test of logic blocks in FPGAs (Finally, A free lunch: BIST without overhead!)," in *Proceedings of IEEE VLSI Test Symposium*, 1996, pp. 387-392.
17. C. Stroud, E. Lee, S. Konala, and M. Abramovici, "Selecting built-in self-test configurations for field programmable gate arrays," in *Proceedings of IEEE Automatic Test Conference*, 1996, pp. 29-35.
18. S. K. Lu and C. W. Wu, "A novel approach to testing LUT-based FPGAs," in *Proceedings of International Symposium on Circuits and Systems (ISCAS)*, Vol. 2, 1999, pp. 69-72.

19. S.-K. Lu, J. S. Sye, and C.-W. Wu, "Testing configurable LUT-based FPGAs," in *Proceedings of 10th VLSI/CAD Workshop*, 1999, pp. 170-174.
20. C. F. Wu and C. W. Wu, "Testing function units of dynamic reconfigurable FPGAs," in *Proceedings of Ninth VLSI/CAD Symposium*, 1998, pp. 189-192.
21. Programmable Gate Array Data Book, Xilinx, Inc., San Jose, CA, 1991.

Shyue-Kung Lu (§f¾Ç©[) received his Ph.D. (1995) in electrical engineering from the National Taiwan University. From 1995 Feb. to 1998 Aug., he was an associate professor in the Department of Electrical Engineering at Lunghwa Junior College of Technology and Commerce. Since then, he has been with the Department of Electronic Engineering at Fu Jen Catholic University, where he is an associate professor. His research interests include the areas of VLSI testing and fault tolerant computing.

Jen-Sheng Shih (¥v®¶¥Í) is a postgraduate of electronic engineering at Fu Jen Catholic University, Taiwan. Mr. Shih received a diploma in electronic engineering from Tungfang Junior College of Technology, Taiwan, in 1993, and the Bachelor's degree in electronic engineering from National Taipei University of Technology, Taiwan, in 1998. He conducts his master's thesis researh on FPGA testing under the supervision of S.-K. Lu.