VLSI DESIGN
2000, Vol. 10, No. 3, pp. 321–333
Reprints available directly from the publisher
Photocopying permitted by license only

Testing and Diagnosing Dynamic Reconfigurable FPGA

CHI-FENG WU and CHENG-WEN WU*

Lab. for Reliable Computing (Rm. 807), Department of Electrical Engineering, National Tsing Hua University, Hsinchu, Taiwan 30013, R.O.C.

(Received 1 February 1999; In final form 1 October 1999)

Dynamic reconfigurable field-programmable logic arrays (FPGAs) are receiving notable attention because of their much shorter reconfiguration time as compared with traditional FPGAs. The short reconfiguration time is vital to applications such as reconfigurable computing and emulation. We show in this paper that testing and diagnosis of the FPGA also can take advantage of its dynamic reconfigurability. We first propose an efficient methodology for testing the interconnects of the FPGA, then present several universal test and diagnosis approaches which cover all functional units of the FPGA. Experimental results show that our approach significantly reduces the testing time, without additional cost for diagnosis.

Keywords: Diagnosis, digital testing, FPGA testing, dynamic reconfigurable FPGA, fault tolerance, universal testing

1. INTRODUCTION

With the advent of deep-submicron VLSI technology, system-on-a-chip is no longer a dream. However, as the integration density and design complexity of system chips keep increasing, design verification is more and more difficult. Emulation and rapid prototyping by field programmable gate arrays (FPGAs) are now widely used to speed up the verification process. They also are used in some first-generation products that need to get into the market soon. In addition to prototyping and emulation, the in-system reprogrammable feature of dynamic reconfigurable FPGAs has made them a natural platform for reconfigurable computing or custom computing.

A typical RAM-based FPGA consists of an array of function units and interconnect channels/matrices, as shown in Figure 1. The function unit and interconnect switches are programmable, *i.e.*, they can be configured to perform different logic functions. Configuration data generated by software tools need to be downloaded to the control memory of the FPGA before it can be used as a

*Address for correspondence: Department of Electrical Engineering, National Tsing Hua University, 101, Sec. 2, Kuang Fu Rd., Hsinchu, Taiwan 30013, ROC. Tel.: +886 3 573-1154, Fax: +886 3 571-5971, e-mail: cww@ee.nthu.edu.tw
From now to late February of 2000: Dr. Cheng-Wen Wu, Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106, USA. Tel.: +1 805 893-3614, Fax: +1 805 893-5440, e-mail: cww@bigbend.ece.ucsb.edu

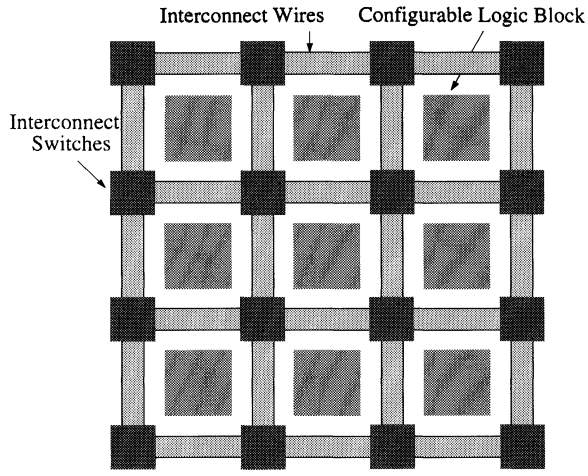

FIGURE 1 General FPGA architecture.

chip designed to the specified function. RAM-based FPGAs can be reprogrammed for virtually unlimited times.

FPGAs can be one time programmable, boot-up configurable, or dynamic reconfigurable. A one time programmable FPGA normally stores the configuration data in a built-in non-volatile memory such as an EEPROM; and a boot-up configurable FPGA normally stores the configuration data in a RAM (configuration bit-stream loading is required for each system boot-up). A dynamic reconfigurable FPGA actually is a special type of boot-up configurable FPGA – its configuration memory can be partially reconfigured: a section of the device can be reconfigured without disturbing circuits already configured in other sections. One time programmable FPGA and boot-up configurable FPGA have been widely used for hardware prototyping. The enhanced programmability of dynamic reconfigurable FPGA makes it even more suitable for emulation and reconfigurable computing [1].

FPGA testing is not trivial. Unfortunately, the FPGA manufacturer is not the only one concerned about its testing. Often, the user needs to do an incoming test to reduce the overall cost of system test. FPGA testing can be done in two ways: testing the unprogrammed FPGA and testing the programmed FPGA. The latter is normally done by a user with test patterns generated for the target circuit configured into the FPGA; however, such user patterns are not efficient even for faults in the configured circuit because of the technology mapping problem [2]. An unprogrammed FPGA can realize a huge amount of different functions, so testing all possible configurations to verify the correctness of the FPGA is not feasible. However, by proper fault modeling and careful selection of configurations, the FPGA can be tested efficiently. A test sequence that fully test the FPGA for the target faults without exercising all possible configurations (*i.e.*, with only a small amount of test configurations) is called a *universal test* [3]. It is universal because it has nothing to do with the target circuit. Note that a universal test still requires a small number of different *test configurations* (*TC*s) and their corresponding *test patterns* (*TP*s). *TC* generation is a very time-consuming process; moreover, *TC* downloading occupies most of the testing time, *i.e.*, time $(TC) \gg \text{time}(TP)$ for each *TC*. To speed up the universal testing process, we must reduce the total number of *TC*s while still able to cover all target faults in the programmable resources of the FPGA, *i.e.*, function units and interconnects.

So far the reported works in FPGA testing are all for boot-up configurable FPGAs, including testing and diagnosis for LUTs [3–5], interconnect testing [6–9], array approaches for testing CLBs in FPGA [10–12], and BIST-based approaches [13–15]. These approaches can be applied to dynamic reconfigurable FPGAs if their architectures are similar, especially for interconnect testing. However, approaches for testing LUTs or CLBs in RAM-based FPGAs are not suitable for dynamic reconfigurable FPGAs because the architectures of their function units are different. Moreover, previous approaches do not take advantage of the dynamic reconfiguration capability during the testing process.

In this paper we focus on testing and diagnosis of dynamic reconfigurable FPGAs. The basic idea is configuring the FPGA into an easily testable array and apply test patterns *via* appropriate

interconnect configurations. Covering all resources by a minimal number test configurations is the goal. Also, using one part of the FPGA to help test other parts is usually helpful. We take advantage of the enhanced programmability of dynamic reconfigurable FPGA and propose universal test approaches that take only a few milli-seconds for testing a typical FPGA. We use a commercial dynamic reconfigurable FPGA, the Xilinx XC6200 [16], as an example for discussing our test methodology. We first introduce the architecture of XC6200, then define the fault models and test patterns for its function units and interconnects. We present in detail how a small amount of test configurations and test patterns can be derived for the interconnects and function units. We also propose universal test and diagnosis approaches for dynamic reconfigurable FPGAs. Our approaches significantly reduce the testing time, and concurrently provide diagnosis capability for faulty function units.

2. XC6200 ARCHITECTURE

The function unit of XC6200 is multiplexer-based, as depicted in Figure 2. The multiplexers are controlled by the configuration memory which is not shown. The function unit can be configured as any two-input logic gate, buffer, inverter, 2-to-1 multiplexer, or any of these in addition to a D-type flip-flop (DFF). There are several ways to configure the DFF. Figure 3 shows three sequential modes of the function unit of XC6200 using different configurations, where the DFF in the rightmost one is said to be in *protected mode*, which makes the DFF accessible only by the programming interface.

The function unit as well as its surrounding interconnect switches (multiplexers) are called the basic cell (BC), as shown in Figure 4. A large array of cells are organized in the form of "sea of gates" at the lowest hierarchy. A 4×4 array of BCs are grouped into a block, which has additional length-4 interconnects and corresponding

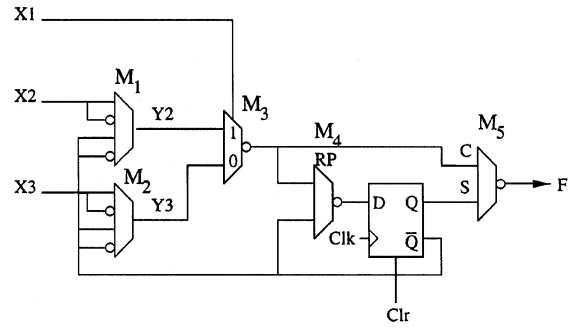

FIGURE 2 XC6200 function unit.

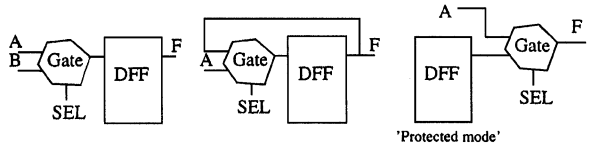

FIGURE 3 Sequential modes of the function units.

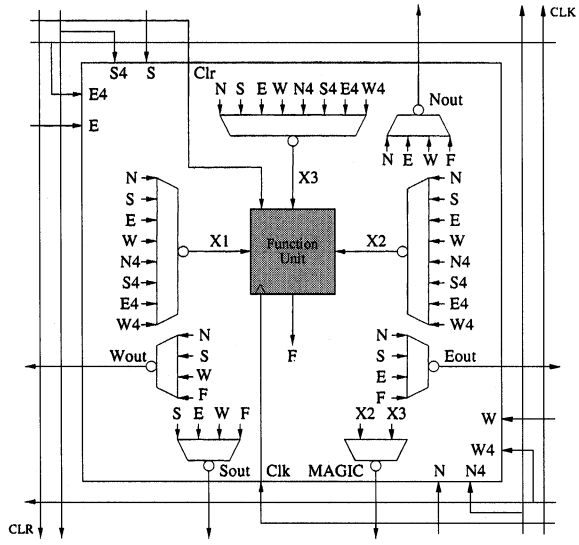

FIGURE 4 XC6216 basic cell.

switches. The hierarchy continues by forming a 16×16 array of BCs (4×4 array of blocks), 64×64 array of BCs, *etc.*, as shown in Figure 5. The length-4 wire can be used to connect to the inputs of other cells in the block, to neighboring blocks, or even to the length-16 wires if it is on the boundary of a 16×16 array. Similarly, the length-16

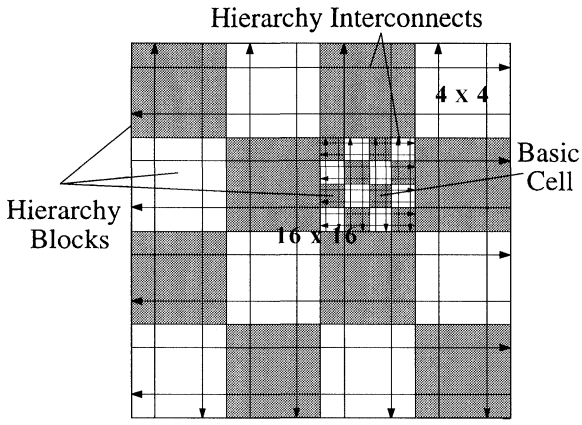

FIGURE 5 XC6200 hierarchy.

wire can be used to connect to the boundary switches of blocks, to neighboring 16×16 arrays, or even to the length-64 wires if it is on the boundary of a 64×64 array. Higher level wires provide efficient long-distance or global routing. For example, the XC6216 FPGA chip is composed of a 64×64 array of BCs. The configuration memories of XC6200 are SRAMs. This allows fast dynamic reconfiguring of function units and interconnect switches. Its full and partial context switching capability is ideal for reconfigurable computing.

3. FAULT MODELS AND TEST PATTERNS

Multiplexer is the elementary component of XC6200, so we propose fault models and test patterns for multiplexer first. A multiplexer is a group of switches which forward exactly one of the inputs directly to the output according to the configuration of switches: the switch for the selected input is on and all others are off. In FPGA, multiplexer control inputs come from the configuration memory. Functionally, the multiplexer can be viewed as a set of configurable switches as shown in Figure 6. In this work, we assume that (1) if all switches are off (open), the multiplexer output is either stuck-at-1 or stuck-at-0, and

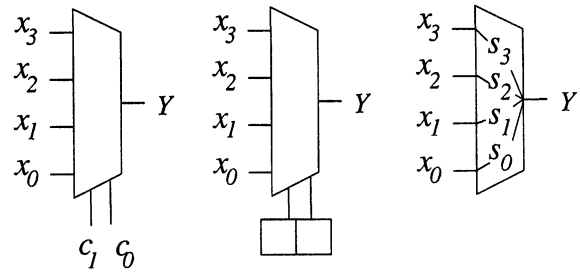

FIGURE 6 Multiplexer functional model.

(2) if two switches are on (closed) simultaneously, the multiplexer output is equivalent to the wired-OR of the two selected inputs. The second assumption is only for ease of discussion. It does not affect the result of our test methodology if it is wired-AND instead of wired-OR.

We consider switch stuck-on faults and stuck-off faults, line bridging faults, and line stuck-at faults as our basic fault models. However, in our case line stuck-at faults are covered by switch stuck-on/off faults. For example, to detect a switch stuck-off or stuck-on fault (assuming CMOS circuits), we must trigger a transition on the corresponding data line. Obviously, then, it also detects stuck-at faults on the input and output data lines. A stuck-at fault on a control input line (also equivalent to a stuck-at fault in the configuration memory) results in multiple stuck-on/off faults. For example, in Figure 6, when c_0 has a stuck-at-0 fault, it is equivalent to a stuck-off fault at s_1 and a stuck-on fault at s_0 if c_1 is 0, or a stuck-off fault at s_3 and a stuck-on fault at s_2 if c_1 is 1. All such cases are detectable by testing all switch stuck-on and stuck-off faults. We obtain the following theorem.

THEOREM 1 *A test which detects all switch stuck-on and stuck-off faults of a multiplexer also detects stuck-at faults on its I/O nets.*

We now propose a test called *MP* for detecting switch stuck-on and stuck-off faults as well as line bridging faults in the multiplexer. By Theorem 1, all target faults will be covered. Consider a multiplexer with data inputs $X = \{x_{k-1}, \ldots, x_1, x_0\}$, switches $S = \{s_{k-1}, \ldots, s_1, s_0\}$, and output $Y = x_c$. Let the configuration input (*i.e.*, switch

control input) be c, where c is a binary number and $0 \leq c < k-1$, then in the fault-free case s_i is on if $i=c$, and it is off otherwise. To test the multiplexer for switch stuck-on and stuck-off faults, we define two test patterns (X vectors) for any configuration vector c:

$$MP_c^0 = \langle mp_{k-1}^0, \ldots, mp_1^0, mp_0^0 \rangle;$$
$$MP_c^1 = \langle mp_{k-1}^1, \ldots, mp_1^1, mp_0^1 \rangle;$$

where $mp_i^0 = 0$ if $i=c$, and $mp_i^0 = 1$ otherwise. Also, $mp_i^1 = \overline{mp_i^0}$. For ease of discussion, we let

$$MP^1 = \{MP_{k-1}^1, \ldots, MP_1^1, MP_0^1\},$$
$$MP^0 = \{MP_{k-1}^0, \ldots, MP_1^0, MP_0^0\},$$

and $MP = MP^1 \cup MP^0$, $MP_c = MP_c^1 \cup MP_c^0$. For example, $MP_3 = MP_3^0 \cup MP_3^1 = \{0111, 1000\}$.

It is obvious that MP activates all switch stuck-on and stuck-off faults, and any fault effect can be observed from the output Y. For example, with $c=0$, MP_0^1 and MP_0^0 together activate the stuck-off fault of s_0, since if the switch is always off then it will fail to transmit either 0 or 1. Also, MP_0^0 activates the stuck-on faults of all switches except s_0, because each of these faults results in a faulty output value (*i.e.*, 1) according to the second assumption of the multiplexer model mentioned above. Note that when the wired-AND logic is assumed instead of wired-OR, MP still activates all stuck-on and stuck-off faults, though stuck-on faults will be activated by MP^1 instead of MP^0.

Bridging (short) faults on input nets of a multiplexer are covered by MP^0 if the fault behavior is equivalent to wired-OR logic, or by MP^1 if wired-AND is assumed. The detection of bridging faults on multiplexer inputs is an important feature of MP because all interconnect switches in the XC6200 series FPGAs are implemented by multiplexers. Detecting bridging faults of the multiplexer inputs implies detecting bridging faults of the interconnect wires.

In summary, to test a multiplexer, we turn on the switches one by one and apply the corresponding MP (see Fig. 7), which covers stuck-on and stuck-off faults of the switches, stuck-at faults of the I/O nets, and bridging faults of the data input nets. Note that although we assume single faults, most multiple faults can also be detected. We will discuss this later.

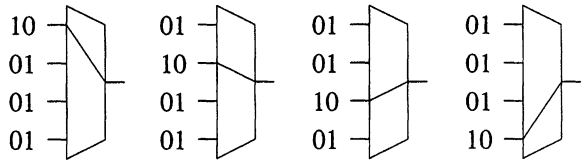

FIGURE 7 Test patterns for a 4-to-1 multiplexer.

4. TESTING THE BASIC INTERCONNECTS

Basic interconnects are implemented by four 4-input multiplexers in the BC, as shown in Figure 8. Parallel testing of these multiplexers is achieved by three TCs, as shown in Figure 9 [17]. In the figure, we show only a 2×2 array for clarity. It can be directly extended to any $N \times N$ array and tested with the same approach.

Multiplexers whose outputs are Nout, Wout, Sout, and Eout are denoted as M_N, M_W, M_S, and M_E, respectively. In the test configuration $TC = (c_1, c_2, c_3, c_4)$, c_1 defines the switch control inputs for M_N, c_2 for M_W, c_3 for M_S, and c_4 for M_E, respectively. For example, if $TC=(E, S, W, N)$, it means that switches E, S, W, and N are turned on in M_N, M_W, M_S, and M_E, respectively. Also, the orthogonal test configuration as shown in Figure 9 is $TC_o=(N, W, S, E)$.

The test pattern is denoted as $TP=\langle p_S, p_E, p_N, p_W \rangle$, where p_S is the two-bit south-bound test sequence, p_E the east-bound test sequence, p_N the north-bound test sequence, and p_W the west-bound test sequence, respectively. According to the test strategy presented in the previous section, applying MP^0 and MP^1 to multiplexers in parallel is our goal. For the orthogonal test configuration, we apply the two-pattern tests $TP_o^0 = \langle 01, 10, 10, 01 \rangle$ and $TP_o^1 = \langle 10, 01, 01, 10 \rangle$ to achieve this goal. Note that F is the output of the function unit

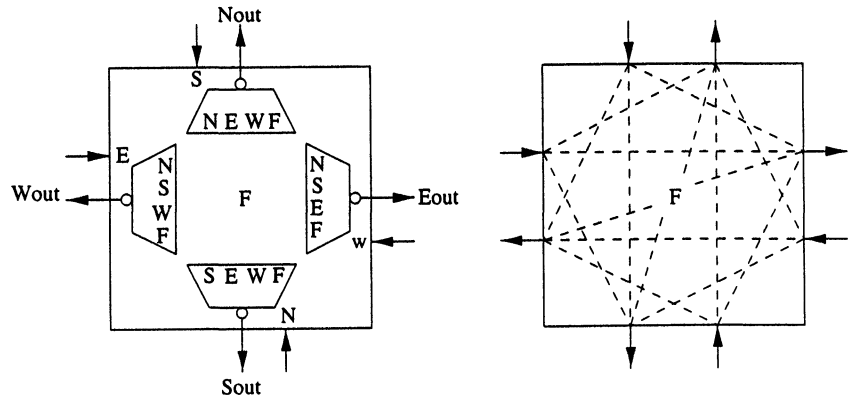

FIGURE 8 Basic routing switches in BC.

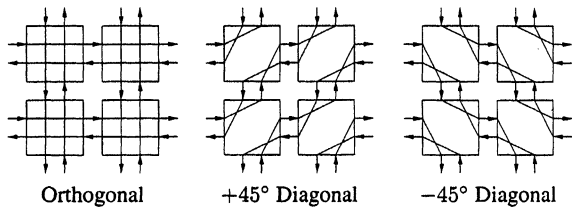

FIGURE 9 *TC*s for basic interconnects.

which is not shown for simplicity. By proper hierarchical routing and configuration of the function unit as a buffer or an inverter, the required value of F in each cell can be assigned. As a result, TP_o^1 and TP_o^0 deliver MP^1 or MP^0 to all cells in parallel with two sets of F values as shown in Figures 10 and 11, respectively. In these figures we show symbolic maps on the right, where the triangles represent the multiplexers at the corresponding locations in the cells. Inside each triangle, there is a circle if MP^0 is applied to the corresponding multiplexer, and a cross ($\times$) if MP^1 is applied instead. Combining these maps we see that MP is successfully applied to each and every multiplexer. Similarly, the $+45°$ diagonal TC is $TC_{d+} = (E, S, W, N)$, and its two-pattern tests are $TP_{d+}^0 = \langle 00, 00, 11, 11\rangle$ and $TP_{d+}^1 = \langle 11, 11, 00, 00\rangle$. For the $-45°$ diagonal TC, $TC_{d-} = (W, N, E, S)$,

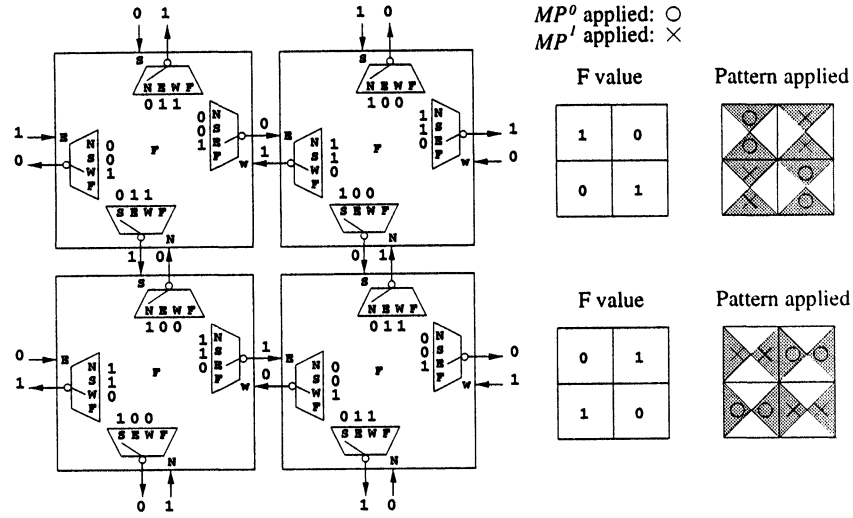

FIGURE 10 Orthogonal configuration with TP_o^0.

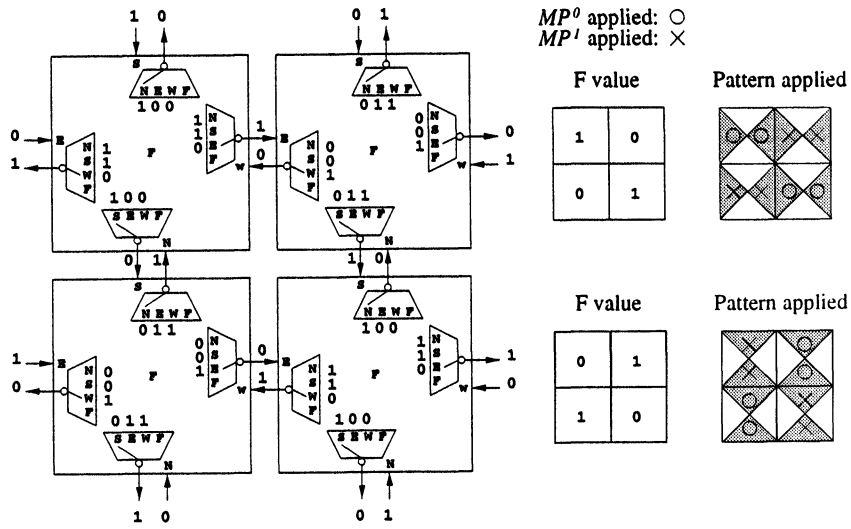

FIGURE 11 Orthogonal configuration with TP_o^1.

the two-pattern tests are $TP_{d-}^0 = \langle 00, 11, 11, 00 \rangle$ and $TP_{d-}^1 = \langle 11, 00, 00, 11 \rangle$.

All stuck-on and stuck-off faults of the basic interconnect switches are covered except the stuck-off faults of the F switches. Testing the F switches requires four configurations: TC_{F_N}, TC_{F_W}, TC_{F_S}, and TC_{F_E}, where, *e.g.*, TC_{F_W} is illustrated in Figure 12. It is clear that if w receives $\langle 01 \rangle$ then the F switch stuck-off fault in M_W is detected. The other three F switch stuck-off faults are covered in a similar way.

In any of the orthogonal, 2 diagonal, and $4F$ configurations, fault effects are propagated in the respective directions to the primary outputs, so given their corresponding test patterns, the $7TC$s test all stuck-on and stuck-off faults in M_S, M_E, M_N, and M_W. By Theorem 1, all multiplexer I/O stuck-at faults are covered. The test MP also covers bridging faults between data inputs lines. Bridging faults between adjacent interconnect wires are detected by the orthogonal TC and the corresponding patterns, which guarantees that adjacent wires carry complementary values during test. In summary, the basic interconnects are fully tested by $7TC$s.

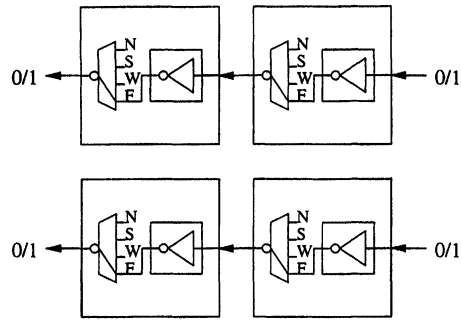

FIGURE 12 TC for F switch stuck-off fault in M_W.

5. TESTING THE FUNCTION UNIT

Testing the function unit by applying test patterns to each and every configurable logic function requires as many TCs as the number of functions. There are 24 combinational functions, each having 3 additional sequential modes [18], so a total of 96 TCs would be required just to test a function unit. However, further investigation shows that the function unit can be fully tested with much fewer TCs [19].

The function unit consists of five multiplexers and a DFF, as shown in Figure 2. Note that MP is not suited to $M1$ and $M2$, which have complementary inputs. To test $M1$ and $M2$, we expand the MP elements. In Table I, the original MP_1^0 is

TABLE I Example of expanded MP_1

Original	Expanded	Selected
$MP_1^0 = 1101$	X101	0101
	1X01	1001
$MP_1^1 = 0010$	X010	1010
	0X10	0110

expanded to two patterns, X101 and 1X01, where the 'X' stands for a don't-care. Test patterns are generated by assigning appropriate values to the don't-cares. The expanded MP has the same fault detection capability as MP.

Test patterns for the function unit, which is a sequential circuit, must be ordered correctly. For example, the test sequence for s_1 of $M1$ is shown in Table II, where the control inputs are $c_{M1}c_{M2}c_{M4}c_{M5} = 1011$. With careful selection of the function unit inputs (*i.e.*, X_1, X_2, and X_3) and the DFF value, the expanded MP of Table I is successfully applied. Note that although X_2 and X_3 are assigned to an identical value, $M1$ and $M2$ are actually configured to transmit complementary values.

To test s_0 of $M1$, an additional pattern is required to invert the DFF value, as shown in Table III, where the control inputs are $c_{M1}c_{M2}c_{M4}c_{M5} = 0111$. The p_2 pattern reset X_1 to 0 to transmit the $\overline{Q}$ value. Upon the application of the next pattern, the DFF value is inverted. As a result, the expanded MP is successfully applied to $M1$. Testing s_2 and s_3 of $M1$ are similar

TABLE II Testing s_1 of $M1$ in the function unit

Test pattern	$X_1X_2X_3$	$\overline{Q}$	$M1$ inputs
p_0	100	1	0110
p_1	100	0	0101
p_2	111	1	1010
p_3	111	0	1001

TABLE III Testing s_0 of $M1$ in the function unit

Test pattern	$X_1X_2X_3$	$\overline{Q}$	$M1$ inputs
p_0	100	1	0110
p_1	111	1	1010
p_2	011	1	1010
p_3	100	0	0101
p_4	111	0	1001

TABLE IV Testing s_0 of $M4$ in the function unit

Test pattern	TC	$X_1X_2X_3$	$\overline{Q}$	$M4$
p_0	0001	111	1	01
p_1	0011	111	1	01
p_2	0001	000	0	10

to testing s_1, and $M2$ can be tested in a similar way as for $M1$.

When we test $M1$ and $M2$, some switch faults in other multiplexers are automatically covered, including all switches of $M3$ and the s_1 switches of $M4$ and $M5$. We now consider testing the rest of the switches, *i.e.*, the s_0 switches of $M4$ and $M5$. Multiplexer $M4$ is also called the *register-protecting* (*RP*) multiplexer. When *RP* is configured to 0, the DFF is in *protected* mode and does not change value regardless of the inputs of the function unit. To test s_0 of $M4$, two *TC*s are required to complement the DFF value. The test configurations (for $M4$) and patterns are listed in Table IV. Before applying p_1, we configure $M4$ to leave the protected mode so that we can invert the DFF value. As a result, MP can successfully be applied to $M4$. This test also covers s_0 of $M5$. In summary, the function unit can be fully tested by 11 *TC*s instead of 96.

6. TESTING AND DIAGNOSIS OF THE FPGA

Although the number of *TC*s to test a single function unit is only 11 for XC6200, testing all function units in the FPGA one by one is not acceptable because that would require tens of thousands of *TC*s. We will show how they can be tested in parallel, requiring only a small number of *TC*s.

We first define the notation. Let the array size be $N \times N$; the number of required *TC*s to test a single function unit be f_c; the average number of test patterns associated with a *TC* for the function unit be f_{ap}; the time for programming interconnects of the FPGA be t_{int}; the time for

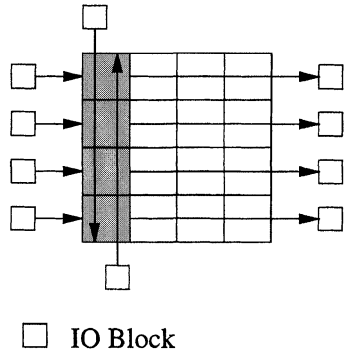

FIGURE 13 Brute-force parallel approach.

dynamically reconfiguring a function unit be t_{fr}; the time for programming the whole FPGA (*i.e.*, downloading a complete *TC* to the FPGA) be t_{TC}, where $t_{TC} = t_{int} + N^2 t_{fr}$; the wire delay of the nearest-neighbor interconnect be t_{wd}; the function unit delay be t_{fd}; and the cell delay be t_{st}, where $t_{st} = t_{fd} + t_{wd}$.

A simplest parallel test approach is to test a row or a column of function units at a time, as shown in Figure 13, which is called the brute-force parallel approach. The two vertical wires in the figure are meant to be global interconnects which deliver test patterns to all BCs under test. The interconnects actually involves wires from the highest level to the lowest level. We use this to represent global nets for simplicity. The brute-force approach is not good enough because it still requires Nf_c *TC*s. The *TC* count grows with the array size, and is not acceptable for large arrays. We propose better approaches below.

6.1. Two-phase Parallel (TPP) Approach

The Reed–Muller propagation chain (RMPC), which is also called the *collector row* in the Reed–Muller canonic network [20], can be used for parallel testing of the XC6200 function units [19]. As shown in Figure 14, the RMPC receives many identical inputs (f) and generates the output Y in the fault-free case. Any single fault at the inputs can automatically be propagated to the output Y, *i.e.*, the value of Y changes given any single input

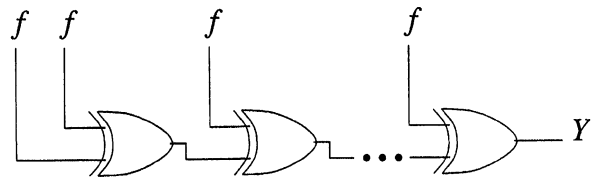

FIGURE 14 Reed–Muller propagation chain.

fault. With RMPCs, multiple function units can be tested simultaneously by the configurations as shown in Figure 15. When the function units in the odd rows are under test, function units in the even rows are configured as RMPCs to propagate possible fault effects. Likewise, when the even rows are under test, the odd rows are configured as RMPCs.

Fault location (diagnosis) of the function units can be done if, in addition to the row-wise configurations, similar column-wise configurations are included to form a 2D addressing of the faulty unit. Apparently any single faulty function unit can be located by using only four *TC*s. This complete test and diagnosis approach requires $2(f_c + k)$ *TC*s, where k is the number of detected faults. The weakness of this approach is that we are unable to detect an even number of faulty units in the same row.

6.2. Dynamic Serial (DS) Approach

The dynamic reconfiguration feature of the FPGA not only increases its programmability but also its testability and diagnosability. Here we propose a new testing and diagnosis approach called the dynamic serial (DS) approach. We first link all function units into a chain, as shown in Figure 16, where all function units are configured to be in the bypass mode (*i.e.*, as buffers). After testing the integrity of the chain in the bypass mode, we test each function unit by its f_c *TC*s and the corresponding patterns, then configure it back to the bypass mode. We repeat the procedure and test the subsequent function units, and continue until all function units have been tested, as shown in Figure 17. Note that when

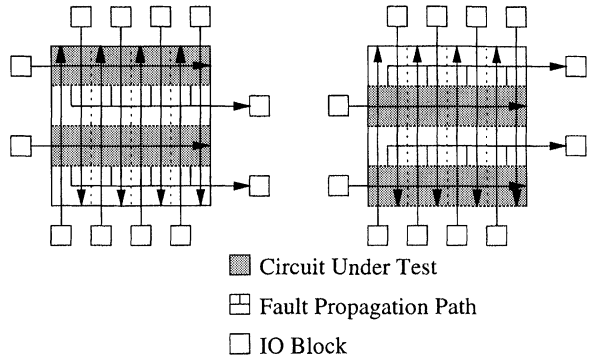

FIGURE 15 Two-phase parallel testing of the function units.

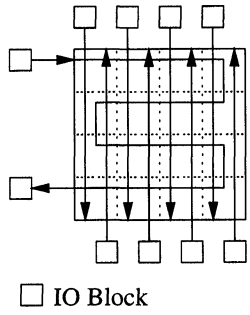

FIGURE 16 Dynamic serial approach.

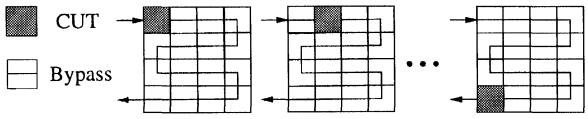

FIGURE 17 Dynamic serial testing procedure.

we test a specific function unit, its configuration data is down-loaded to the FPGA dynamically, *i.e.*, the configuration of other function units remain unchanged. Therefore, the total configuration time is $t_{TC}+N^2(f_c+1)t_{fd}$, which is much shorter than that for the two-phase parallel approach, especially for a large N.

Although this approach takes advantage of the dynamic reconfiguration feature of the FPGA and reduces the configuration time, the delay time of the serial path (*i.e.*, the application time for a test pattern) still increases with the array size N^2. To solve the problem, we propose an improved approach called the dynamic serial-parallel (DSP) approach, which is discussed next. Note that fault diagnosis is automatic in both approaches.

6.3. Dynamic Serial-parallel (DSP) Approach

The idea is simple. We partition the original single serial path (the chain of all function units) in the DS approach into multiple paths (still covering all function units) to reduce the path delay in large arrays, as shown in Figures 18 and 19. For the shortest path delay, we can configure the paths so that each of them consists only of a single row or column of function units. This approach maintains the short test configuration

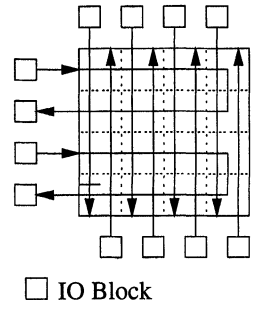

FIGURE 18 Dynamic serial-parallel approach.

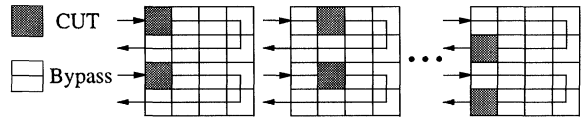

FIGURE 19 Dynamic serial-parallel testing procedure.

time as DS, while greatly reduces the test application time.

7. TIME COMPLEXITY AND ANALYSIS

Normally the testing time is dominated by the configuration time. As we have mentioned, our primary objective was to minimize the number of required *TC*s. However, the path delay also should be taken into consideration when we use the dynamic approaches, since test pattern application time is dependent on the path delay. The test time of the TPP, DS, and DSP approaches are shown, respectively, by the following equations:

$$T_{\mathrm{TPP}} = 2(f_c + k)t_{TC} + 2(f_c + k)f_{ap}Nt_{st};$$
$$T_{\mathrm{DS}} = t_{TC} + N^2(f_c + 1)t_{fd} + f_c f_{ap}N^4 t_{st};$$
$$T_{\mathrm{DSP}} = t_{TC} + N^2(f_c + 1)t_{fd} + f_c f_{ap}s^2N^2 t_{st};$$

where sN is the length of a path (*i.e.*, $1 \le s \le N$).

In each equation, the first term in the right-hand side represents the test configuration time, and the second term represents the time to apply the test patterns. Take XC6216 as an example, where $N=64$, $f_c=11$, $t_{TC}=0.66$ ms, $t_{fd}=60$ ns, $f_{ap}=3.4$, $t_{st}=3.5$ [16]. Assume $k=1$ for T_{TPP} (*i.e.*, single fault diagnosis), and $s=1$ for T_{DSP} (*i.e.*, one row serial), then the results are, respectively, $T_{\mathrm{TPP}}=15.96$ ms, $T_{\mathrm{DS}}=2.2$ s, and $T_{\mathrm{DSP}}=4.15$ ms. Clearly DSP is the fastest approach in this case. For larger chips, DSP will remain to be the best, and the improvement over TPP will be even more significant, as can be seen from Figure 20. The reason is that in the equation for T_{TPP}, there is a larger and growing coefficient for t_{TC}, which grows linearly with the array size (N^2). The curve for DSP in the figure also shows that the serial path delay does increase the testing time as expected, though with a much lower weight as compared with t_{TC}. However, in DS the test pattern application time is even longer

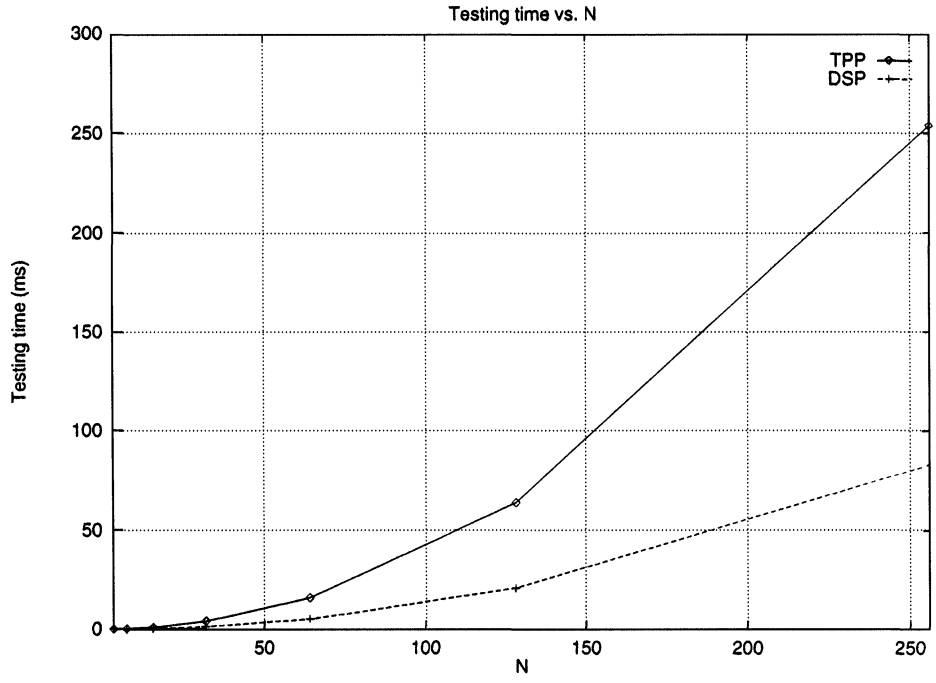

FIGURE 20 Testing time comparison.

than the test configuration time, so it becomes the worst of the three.

From our time complexity analysis, DSP is faster than TPP, and the gap grows with the array size. In practice, DSP is also more flexible than TPP when we take the number of I/O pins into consideration. With DSP, we can trade I/O pins for pattern application time, *e.g.*, we can double the length of the serial path to reduce the number of I/O pins in half. The test configuration time remains the same. However, with TPP, we have to double the test configuration time in order to reduce the same number of I/O pins.

TPP and DSP are both general approaches for dynamic reconfigurable FPGAs, but DSP is more suitable for those with fine-grain reconfiguration capability, while TPP can also be applied to boot-up configurable FPGAs.

8. CONCLUSIONS

FPGA has been widely used in hardware prototyping and emulation, and considered the key hardware component in custom and reconfigurable computing. Testing FPGAs therefore is an important issue to the manufacturers as well as the end users. We have shown that the testing time is dominated by the time to download the test configurations, and have proposed approaches whose primary objective is to minimize the number of test configurations. The experimental results justify the objective that we have aimed at. We also have proposed universal test and diagnosis approaches for dynamic reconfigurable FPGAs, including two dynamic approaches which take advantage of the enhanced programmability (*i.e.*, dynamic partial reconfigurability) of the dynamic reconfigurable FPGAs. Our dynamic serial-parallel approach significantly reduces the testing time, and concurrently provides diagnosis capability for faulty function units. Finally, we have implemented several test configurations with the Xilinx XACT 6000 design kit and have done some experiments on a PCI board. Correct results have been obtained. However, the speed was limited by the interface and the PCI board. The issue should be able to be solved easily by the industry.

References

[1] Buell, D. A., Arnold, J. M. and Kleinfelder, W. J. (1996). *Splash 2: FPGAs in a custom computing machine*. Los Alamitos, CA: IEEE Computer Society Press.

[2] Kwiat, K., Debany, W. and Hariri, S., "Effects of technology mapping on fault-detection coverage in reprogrammable FPGAs", *IEE Proc.-Comput. Digit. Tech.*, **142**, 407–410, Nov., 1995.

[3] Inoue, T., Fujiwara, H., Michinishi, H., Yokohira, T. and Okamoto, T., "Universal test complexity of field-programmable gate arrays", In: *Proc. Fourth Asian Test Symp. (ATS)* (Bangalore), pp. 259–265, Nov., 1995.

[4] Inoue, T., Miyazaki, S. and Fujiwara, H., "Universal fault diagnosis for lookup table FPGAs", *IEEE Design and Test of Computers*, **15**, 39–44, Jan.–Mar., 1998.

[5] Huang, W.-K., Meyer, F. J., Chen, X.-T. and Lombardi, F. (1998). "Testing configurable LUT-based FPGAs", *IEEE Trans. VLSI Systems*, **6**(2), 276–283.

[6] Huang, W.-K. and Lombardi, F. (1996). "An approach for testing programmable/configurable field programmable gate arrays", In: *Proc. 14th IEEE VLSI Test Symp. (VTS)*, pp. 450–455.

[7] Michinishi, H., Yokohira, T., Okamoto, T., Inoue, T. and Fujiwara, H., "A test methodology for interconnect structures of LUT-based FPGAs", In: *Proc. Fifth Asian Test Symp. (ATS)*. (Hsinchu), pp. 68–74, Nov., 1996.

[8] Renovell, M., Figueras, J. and Zorian, Y., "Test of RAM-based FPGA: Methodology and application to the interconnect", In: *Proc. 15th IEEE VLSI Test Symp. (VTS)*, (Monterey), pp. 230–237, Apr., 1997.

[9] Renovell, M., Portal, J. M., Figueras, J. and Zorian, Y., "Testing the interconnect of RAM-based FPGAs", *IEEE Design and Test of Computers*, **15**, Jan.–Mar., 1998.

[10] Huang, W.-K., Chen, X.-T. and Lombardi, F. (1996). "On the diagnosis of programmable interconnect systems: Theory and application", In: *Proc. 14th IEEE VLSI Test Symp. (VTS)*, pp. 204–209.

[11] Huang, W.-K., Zhang, M.-Y., Meyer, F. J. and Lombardi, F., "A XOR-tree based technique for constant testability of configurable FPGAs", In: *Proc. Sixth Asian Test Symp. (ATS)* (Akita), pp. 248–253, Nov., 1997.

[12] Inoue, T., Miyazaki, S. and Fujiwara, H., "On the complexity of universal fault diagnosis for look-up table FPGAs", In: *Proc. Sixth Asian Test Symp. (ATS)* (Akita), pp. 276–281, Nov., 1997.

[13] Stroud, C., Konala, S., Chen, P. and Abramovici, M. (1996). "Built-in self-test for programmable logic blocks in FPGAs (Finally, a free lunch: BIST without overhead!)", In: *Proc. 14th IEEE VLSI Test Symp. (VTS)*, pp. 387–392.

[14] Stroud, C., Lee, E., Konala, S. and Abramovici, M. (1996). "Using ILA testing for BIST in FPGAs", In: *Proc. Int. Test Conf. (ITC)*, pp. 68–75.

[15] Stroud, C., Lee, E. and Abramovici, M. (1997). "BIST-based diagnostics of FPGA logic blocks", In: *Proc. Int. Test Conf. (ITC)*, pp. 539–547.

[16] Xilinx, *XC6200 Field Programmable Gate Arrays*. San Jose, California: Xilinx, Inc., Apr., 1997.
[17] Wu, C.-F. and Wu, C.-W., "Testing interconnects of dynamic reconfigurable FPGAs", In: *Proc. Asia and South Pacific Design Automation Conf. (ASP-DAC)* (Hong Kong), pp. 279–282, Jan. 1999.
[18] Xilinx (1996). *The Programmable Gate Array Data Book*. San Jose, California: Xilinx, Inc.
[19] Wu, C.-F. and Wu, C.-W., "Testing function units of dynamic reconfigurable FPGAs", In: *Proc. 9th VLSI Design/CAD Symp.* (Nantou), pp. 189–192, Aug., 1998.
[20] Saluja, K. K. and Reddy, S. M., "Fault detecting test sets,for Reed–Muller canonic network", *IEEE Trans. Computers*, **24**, 995–998, Nov., 1975.

Authors' Biographies

Chi-Feng Wu received the BSEE and MSEE in Electrical Engineering from National Tsing Hua University. He is currently working toward PhD. His research interests are testing for programmable logic devices (including FPGAs and CPLDs), memory testing, and memory fault simulation. Mr. Wu is a student member of IEEE.

Cheng-Wen Wu received the BSEE degree from National Taiwan University and the MS and PhD degrees in ECE from UC, Santa Barbara. He is a, professor of Electrical Engineering at National Tsing Hua University, Taiwan, and a Guest Editor of the Journal of Information Science and Engineering, Special Issue on VLSI Testing. Dr. Wu was the Program Chair of the IEEE Fifth Asian Test Symposium, and is the General Chair of the Ninth ATS. He received the Distinguished Teaching Award from NTHU and the Outstanding Electrical Engineering Professor Award from the Chinese Institute of Electrical Engineers. He is interested in VLSI design and test. He is a member of CIEE and a senior member of IEEE.