Universal Test Complexity of Field-Programmable Gate Arrays

Tomoo Inoue, Hideo Fujiwara, Hiroyuki Michinishi, Tokumi Yokohira, and Takuji Okamoto

Abstract— **A field-programmable gate array (FPGA) can implement arbitrary logic circuits in the field. In this paper, we consider *universal test* such that when applied to an unprogrammed FPGA, it ensures that all the corresponding programmed logic circuits on the FPGA are fault-free. We focus on testing for look-up tables in FPGAs, and present two types of programming schemes; *sequential loading* and *random access loading*. Then we show test procedures for the FPGAs with these programming schemes and their test complexities. In order to make the test complexity for FPGAs independent of the array size of the FPGAs, we propose a programming scheme called *block-sliced loading*, which makes FPGAs *C-testable*.**

Keywords— **C-testable, field-programmable gate array (FPGA), programming scheme, test complexity, test procedure, universal test.**

I. Introduction

FIELD-PROGRAMMABLE gate arrays (FPGAs) are digital devices that can implement logic circuits required by users in the field [1], [2]. Because of their short turnaround time, low manufacturing cost and programmability in the field, there has been an increasing interest in system prototyping and system reconfiguration using FPGAs. There are many different architectures of FPGAs driven by different programming technologies. One important class is the SRAM-based FPGAs (e.g. Xilinx [1], [2], [3]), also called the look-up table FPGAs, which can be reprogrammed any number of times. A novel feature of these FPGAs is that each basic block can implement any logic function that satisfies the I/O constraints of the basic block. The interconnections between the basic blocks consist of metal segments joined by program controlled pass transistors. In this paper, we shall consider look-up table FPGAs.

Testing for FPGAs, as well as conventional digital ICs, is one of the important problems. Several works on testing FPGAs have been reported [4], [5]. Hermann and Hoffmann [4] presented fault models and test generation for one-time programmable FPGAs (e.g. Actel's [1], [2]). Durate and Nicolaidis [5] reported a test methodology for cellular-based FPGAs (e.g. Algotronix's [1], [2]). For reprogrammable FPGAs, two types of testing can be considered; one is testing for unprogrammed FPGAs, and the other is testing for programmed FPGAs. An unprogrammed FPGA can realize many different programmed FPGAs by loading different programs. Therefore, to test the unprogrammed FPGA, we might have to test all the programmed FPGAs obtained from the unprogrammed FPGA. However, it is too time-consuming to test such a large number of programmed FPGAs. In order to resolve this intractable problem, we have to consider alternative approaches to testing for unprogrammed FPGAs.

In this paper, we shall introduce *universal test* such that when applied to a given *un*programmed FPGA, it ensures that *all* programmed FPGAs corresponding to the unprogrammed FPGA are fault-free. Here, we focus on testing for look-up tables in FPGAs. Testing for other components in an FPGA can be considered in the same way as testing for look-up tables. Then we shall present *test complexity* of FPGAs, where test complexity of an FPGA refers to the time required to test the FPGA. We shall present two types of programming schemes; *sequential loading* and *random access loading*, and show that the test complexities of FPGAs with these programming schemes are $O(Nn\log n)$ and $O(Nn)$, respectively, where N is the array size of FPGAs or the number of configurable logic blocks, and n is the size of look-up tables or the number of configuration memory cells for each look-up table. The test complexities of these FPGAs depend on the array size N, and thus they might not be *C-testable* [6]. If we can make FPGAs C-testable, we can considerably reduce the test complexity. Therefore, we shall propose a new programming scheme, called *block-sliced loading*, which makes FPGAs C-testable. The test complexities of test procedures for FPGAs with block-sliced sequential loading and block-sliced random access loading are $O(n\log^2 n + \log^3 n)$ and $O(n + \log n)$, respectively.

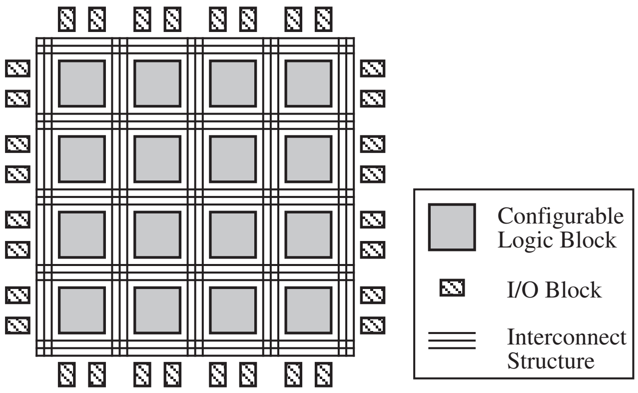


Fig. 1. Architecture of FPGA.

II. Architecture of FPGA

The architecture of field-programmable gate arrays (FPGAs) considered in this paper is illustrated in Fig. 1. An FPGA consists of an array of programmable logic blocks, programmable I/O blocks, and a programmable interconnect structure. Each logic block consists of a single *look-up table* (LUT). These blocks and the interconnect structure are configured by static RAMs called configuration memory cells. This FPGA is referred to as a *look-up table FPGA*.

An LUT implements combinational logic as a $2^k \times 1$ memory composed of configuration memory cells, where k is the num-

T. Inoue and H. Fujiwara are with the Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma, Nara 630-01, Japan. E-mail: inoue@is.aist-nara.ac.jp.

H. Michinishi, T. Yokohira, and T. Okamoto are with the Faculty of Engineering, Okayama University, Okayama 700, Japan.

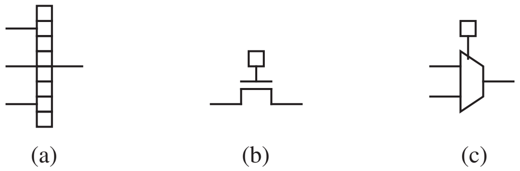


Fig. 2. Configurable components. (a) Look-up table. (b) Pass transistor. (c) Multiplexer.

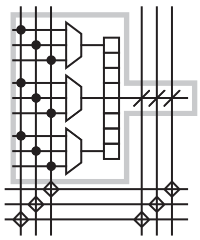


Fig. 3. Configurable logic block.

ber of input lines of the LUT. When an input pattern is applied to an LUT , the LUT selects a configuration memory cell addressed by the input pattern, and the output of the cell provides the value of the function. An LUT can therefore implement any of 2^n functions of its inputs, where $n = 2^k$. When the FPGA is programmed, the memory is loaded with the bit pattern corresponding to the truth table of the function. Fig. 2 (a) shows a block of a three-input LUT.

A pass transistor controlled by a configuration memory cell, as shown in Fig. 2(b), configures a connection of wiring segments in an interconnect structure. The wire segments on each side of the transistor are connected or not, depending on the value in the memory cell. A multiplexer, as shown in Fig. 2(c), also controls a connection of wiring segments. Multiplexers may be of any width, with more configuration memory cells for wider multiplexers. Fig. 3 shows the building blocks from Fig. 2 combined into a configurable logic block (CLB) with wiring. The CLB in Fig. 3 includes a single three-input LUT surrounded by wiring channels. Each wiring channel contains several segments. Segments have connections to the CLB and to each other through pass transistors and multiplexers.

A look-up table FPGA is programmed by loading a *program* composed of a bit sequence into its configuration memory cells. Each bit of the program is stored in the corresponding configuration memory cell, and consequently LUTs and interconnections are configured. Accordingly a logic function or a *configuration* is implemented on the FPGA. The FPGA must include circuitry to load a program. Here we consider two types of programming schemes as follows.

Sequential loading: When an FPGA is programmed, the program is shifted into the FPGA, and each bit of the program is stored in the corresponding configuration memory cell. This type of loading scheme is called *sequential loading*, and an FPGA with this type of loading is called a *sequentially loadable* FPGA (*SL-FPGA*). Whenever an SL-FPGA implements configurations, it loads all configuration memory cells.

Random access loading: Each configuration memory cell is directly addressable. When an FPGA is programmed, each bit is loaded by means of its address, and stored in the corresponding cell. This type of loading scheme is called *random access loading*, and an FPGA with this type of loading is called a *random access loadable FPGA* (*RAL-FPGA*). An RAL-FPGA can implement a configuration by loading only the bits which differ from those of the previous one.

III. Universal Test

We can consider testing for FPGAs as two types of testing; one is testing for unprogrammed FPGAs and the other is testing for programmed FPGAs. As testing for unprogrammed FPGAs, we introduce *universal test* such that when applied to an *un*programmed FPGA, it ensures that *all* the configurations implemented on the corresponding FPGA are fault-free.

An LUT is one of the important and characteristic configurable components in a look-up table FPGA. A k-input LUT consists of 2^k configuration memory cells, and each memory cell has its own address. When an input pattern is applied to the LUT from its k input lines, the LUT *decodes* the input pattern and *reads out* the memory cell corresponding to the input pattern. Hence, an LUT can be considered as a random access memory (RAM). However, since the decoder for writing (or programming) is different from that for reading, conventional methods for testing of RAMs [7], [8] to such LUTs can not be applied.

A. Fault Model

Let k be the number of input lines of an LUT. Let $n = 2^k$ be the number of configuration memory cells of an LUT. Let $A = \{a_0, a_1, \ldots, a_{n-1}\}$ denote a set of input patterns for an LUT, i.e., a set of addresses of configuration memory cells of the LUT. Let $M = \{m_0, m_1, \ldots, m_{n-1}\}$ denote a set of configuration memory cells in an LUT. The decoding function of an LUT can be modeled as a mapping from A to 2^M. The function of a fault-free (i.e. correct) LUT is expressed as $f(a_i) = \{m_i\}$ for $0 \leq i \leq n-1$. Note that the range of the fault-free decoding function f is a singleton for all a_i.

Then, we define four fault models as follows.

Stuck-at fault (SAF): The value that is read out from the memory cell m_i corresponding to input pattern a_i is always either 0 or 1, irrespective of configurations. This type of fault is called a *stuck-at fault (SAF)*. Note that an SAF is independent of the decoding function.

Incorrect access fault (IAF): For some input pattern $a_i \in A$, whenever memory cell m_i is to be accessed by input pattern a_i, another memory cell m_j which does not correspond to a_i is accessed. That is,

$$f(a_i) = \{m_j\} \neq \{m_i\}.$$

This type of fault is called an *incorrect access fault (IAF)*.

Non-access fault (NAF): For some input pattern $a_i \in A$, whenever memory cell m_i is to be accessed by input pattern a_i, no memory cell is accessed. That is,

$$f(a_i) = \phi.$$

This type of fault is called a *non-access fault (NAF)*. The output value for the input pattern a_i becomes the same as the output value before applying the input pattern.

Multiple access fault (MAF): For some input pattern $a_i \in A$, whenever only memory cell m_i is to be accessed by input pattern a_i, more than one memory cells are accessed. That is,

$$|f(a_i)| \geq 2.$$

This type of fault is called a *multiple-access fault (MAF)*. The output value under an MAF is formed by the bitwise OR or AND function, which depends on the device technology, over the memory cells that are accessed by a_i. over the memory cells in the set $f(a_i)$.

Hereafter, we consider universal test for these faults of LUTs provided that there exists only one fault in an FPGA [1].

B. Universal Test Procedure

The universal test for look-up table FPGAs can be performed by repeating implementation of a configuration and application of an input sequence to the configuration alternately. Hence, a *universal test procedure* for an FPGA is represented by a sequence of pairs of a configuration and an input sequence applied to the configuration. Thus, a universal test procedure TP is expressed as

$$TP = \langle (C_1, S_1), (C_2, S_2), \ldots, (C_{n_c}, S_{n_c}) \rangle$$

where C_i is the i-th configuration, S_i is the input sequence which is applied to the i-th configuration C_i, and n_c is the number of configurations in test procedure TP.

When a configuration on an SL-FPGA is changed to another one, the program bits to be loaded into all the configuration memory cells in the FPGA are required. In general, the number of configuration memory cells in an FPGA is large, and hence SL-FPGAs require that the number n_c of configurations in a universal test procedure is small. On the other hand, when a configuration on an RAL-FPGA is changed to another one, only the bits which differ from those of the previous configuration can be loaded. Hence, RAL-FPGAs require that the differences between a configuration and the consecutive one in a universal test procedure are small. Therefore, we have to consider two universal test procedures severally. In the following, we first consider universal test procedures for a single LUT, and next we modify them into those for all the LUTs in an FPGA.

B.1 Universal Test Procedure for Single LUT

First, we consider a universal test procedure for a single LUT. Let k be the number of input lines of an LUT. Let $n = 2^k$ be the number of configuration memory cell of an LUT. For the sake of simplicity, we assume that input pattern a_j for an LUT denotes a binary representation of decimal number j for $0 \leq j \leq n-1$ without loss of generality. Let $b(a_j, l)$ be the l-th bit of pattern a_j. Let $Load(m_j, v)$ denote loading a value v into memory cell m_j. Note that this operation $Load()$ can be skipped if the programming scheme of the FPGA is random access loading and if the value to be loaded into a memory cell is the same as that in the previous configuration. Let $Input(a_j)$ denote applying input pattern a_j to read out the value in the corresponding memory cell m_j.

[1]This assumption is for the sake of simplicity. We can show that the test procedures presented in the rest of paper can detect any multiple fault that consists of SAFs, IAFs, NAFs and MAFs provided that the number of faulty LUTs is at most one in an FPGA.

```
1:  TP1() {
2:       for(i = 1; i <= 2k; i++) { /* Ci: i-th config. */
3:            for(j = 0; j < n; j++) {
4:                 if(i <= k) {
5:                      Load(mj, b(aj,k-i));
6:                 } else {
7:                      Load(mj, ~b(aj,2k-i));
8:                 }
9:            }
10:           for(j = 0; j < n; j++) {
11:                Input(aj);
12:           }
13:           if(i == 1) {
14:                Input(a0);
15:           }
16:      }
17: }
```

Fig. 4. Test procedure: TP_1.

Fig. 4 shows a universal test procedure for a single LUT. This test procedure is called TP_1. As shown in this figure, in test procedure TP_1, memory cell m_j is loaded with the $(k-i)$-th bit of a_j at the i-th configuration C_i for $1 \leq i \leq k$, and is loaded with the complement of the $(2k-i)$-th bit of a_j at the i-th configuration C_i for $k+1 \leq i \leq 2k$. Then, all memory cells are read out by applying all the input patterns for each configuration. At the first configuration C_1, input pattern a_0 is applied not only as the first input pattern but also as the last one to detect an NAF for a_0 (Lines 13 to 15 in Fig. 4).

Let $r(i, j)$ be the output value of the LUT for the j-th input pattern at the i-th configuration. For $0 \leq j \leq n-1$, let $R_1(j)$ and $R_2(j)$ be the sequences obtained by arranging all the output values corresponding to the j-th input patterns at the i-th configurations C_i for $1 \leq i \leq k$ and for $k+1 \leq i \leq 2k$, respectively. That is,

$$\begin{aligned} R_1(j) &= (r(1,j), r(2,j), \ldots, r(k,j)) \\ R_2(j) &= (r(k+1,j), r(k+2,j), \ldots, r(2k,j)) \end{aligned}$$

When the LUT is fault-free, the sequences $R_1(j)$ and $R_2(j)$ have the following properties for $0 \leq j \leq n-1$.

$$R_1(j) = a_j \tag{1}$$

$$\begin{aligned} R_2(j) &= (r(k+1,j), r(k+2,j), \ldots, r(2k,j)) \\ &= \left(\overline{r(1,j)}, \overline{r(2,j)}, \ldots, \overline{r(k,j)}\right) \\ &= \overline{R_1(j)} \\ &= \overline{a_j} \end{aligned} \tag{2}$$

where $\overline{x}$ denotes the bitwise complement of x.

Table I shows an example of the configurations in test procedure TP_1 for the number of input lines $k = 2$.

For universal test procedure TP_1, we have the following lemmas.

TABLE I
EXAMPLE: TP_1 FOR $k = 2$.

memory cell address		C_1	C_2	C_3	C_4
m_0	00	0	0	1	1
m_1	01	0	1	1	0
m_2	10	1	0	0	1
m_3	11	1	1	0	0

Lemma 1: TP_1 can detect any SAF.

Proof: Suppose that the output value from a memory cell m_j is stuck at 0 (1). Output value $r(i, j) = 0$ (1) for all i, and consequently

$$R_1(j) = R_2(j) = 00\ldots0\ (11\ldots1).$$

This is inconsistent with Equations (1) and (2). ■

Lemma 2: TP_1 can detect any IAF.

Proof: Suppose that an input pattern a_j selects a memory cell m_s ($s \neq j$). The sequence $R_1(j)$ is expressed as

$$R_1(j) = a_s.$$

This is inconsistent with Equation (1) because there is no pair of addresses a_s and a_j such that $a_j = a_s$. ■

Lemma 3: TP_1 can detect any NAF.

Proof: Suppose that an input pattern a_j selects no memory cell.

(Case where $j = 0$): At the first configuration C_1, memory cells m_0 and m_{n-1} are loaded with 0 and 1, respectively. Hence, if the LUT is fault-free,

$$r(1,n) \neq r(1,n-1). \tag{3}$$

Memory cell m_0 is to be selected by address a_0 which is the last input pattern of the sequence S_1 at the first configuration C_1 after memory cell m_{n-1} is read out (line 14 in Fig. 4). If a_0 selects no memory cell, $r(1,n) = r(1,n-1)$. This is inconsistent with Equation (3).

(Case where $1 \le j \le n-1$): At the k-th configuration C_k, 0 and 1 are loaded into all memory cells alternately. Hence, if the LUT is fault-free,

$$r(k,l) \neq r(k,l-1) \tag{4}$$

for $1 \le l \le n-1$. If an input pattern a_j selects no memory cell, $r(k, j) = r(k, j-1)$. This is inconsistent with Equation (4). ■

Lemma 4: TP_1 can detect any MAF.

Proof: Suppose that an input pattern a_j selects m_s and m_t ($s \neq t$). From Equations (1) and (2), the sequences $R_1(j)$ and $R_2(j)$ are expressed as

$$\begin{aligned} R_1(j) &= a_s * a_t \text{ and} \\ R_2(j) &= \overline{a_s} * \overline{a_t}, \end{aligned}$$

respectively, where $*$ denotes either bitwise AND or bitwise OR function depending on technology. Hence,

$$R_1(j) \neq \overline{R_2(j)}.$$

```
TP2() {
    if (Outputs under MAFs are formed by AND ) {
        v = 0;
    } else { /* by OR */
        v = 1;
    }
    for (i = 1; i <= n; i++) { /* Ci: i-th config. */
        for (j = 0; j < n; j++) {
            if (j == i - 1) {
                Load(mj, v̄);
            } else {
                Load(mj, v);
            }
        }
        j = i - 1;
        Input(aj);
        if (b(aj,0) == 0) {
            l = j+1;
        else {
            l = j-1;
        }
        Input(al);
    }
}
```

Fig. 5. Test procedure: TP_2.

This is inconsistent with Equation (2). ■

From Lemmas 1 through 4, we have the following theorem.

Theorem 1: Universal test procedure TP_1 can detect any fault in an LUT.

Next we present another universal test procedure for a single LUT, called TP_2. Fig. 5 shows universal test procedure TP_2. In test procedure TP_2, at the i-th configuration, only the $(i-1)$-th memory cell m_{i-1} is loaded with $\overline{v}$ and the others are loaded with v's, where value v is determined by technology. Let $l(j,x)$ denote the configuration number when memory cell m_j that is loaded with x is read out, and it can be expressed as

$$\begin{aligned} l(j,\overline{v}) &= j+1 \\ l(j,v) &= \begin{cases} j+2 & \text{if } j \text{ is even} \\ j & \text{otherwise (odd)} \end{cases} \end{aligned}$$

for all j. When the LUT is fault-free, the output values for test procedure TP_2 are expressed as

$$\begin{aligned} r(i,0) &= \overline{v} \qquad (5) \\ r(i,1) &= v \\ &= \overline{r(i,0)} \qquad (6) \end{aligned}$$

for all i.

Table II shows an example of the configurations in test procedure TP_2 for the number of input lines $k = 2$.

For test procedure TP_2, we have the following lemmas.

Lemma 5: TP_2 can detect any SAF.

Proof: Suppose that the output value that is read out from a memory cell m_j is stuck at v ($\overline{v}$). The output values for input pattern a_j are expressed as

$$r(l(j,\overline{v}),0) = r(l(j,v),1) = v\ \ (\overline{v})$$

TABLE II
EXAMPLE: TP_2 FOR $k=2$.

memory cell	address	configuration C_1	C_2	C_3	C_4
m_0	00	$\overline{v}$	v	v	v
m_1	01	v	$\overline{v}$	v	v
m_2	10	v	v	$\overline{v}$	v
m_3	11	v	v	v	$\overline{v}$

This is inconsistent with Equation (5) or (6). ∎

Lemma 6: TP_2 can detect any IAF.

Proof: Suppose that an input pattern a_j selects a memory cell m_s ($s \neq j$). At the $(j+1)$-th configuration, $m_s = v$. Hence, the output value for input pattern a_j is expressed as

$$r(j+1,0) = v$$

This is inconsistent with Equation (5). ∎

Lemma 7: TP_2 can detect any NAF.

Proof: Suppose that an input pattern a_j selects no memory cell. At the $l(j,v)$-th configuration, input pattern a_j is applied as the second pattern after the value $\overline{v}$ is read out. The previous output value $\overline{v}$ is kept due to the NAF, i.e.,

$$\begin{aligned} r(l(j,v),1) &= r(l(j,v),0) \\ &= \overline{v} \end{aligned}$$

This is inconsistent with Equation (6). ∎

Lemma 8: TP_2 can detect any MAF.

Proof: Suppose that an input pattern a_j selects m_s and m_t ($s \neq t$). The output value for input pattern a_j can be expressed as

$$r(l(j,\overline{v}),0) = d(m_s) * d(m_t)$$

where $d(m_s)$ is the value that is loaded into m_s, and $*$ denotes either AND or OR function depending on technology. For each configuration, there is not more than two memory cells that are loaded with $\overline{v}$'s. Hence,

$$d(m_s) * d(m_t) = v$$

for any pair of m_s and m_t such that $s \neq t$. Thus,

$$r(l(j,\overline{v}),0) = v.$$

This is inconsistent with Equation (5). ∎

From Lemmas 5 through 8, we have the following theorem.

Theorem 2: Universal test procedure TP_2 can detect any fault in an LUT.

B.2 Universal Test Procedure for All LUTs

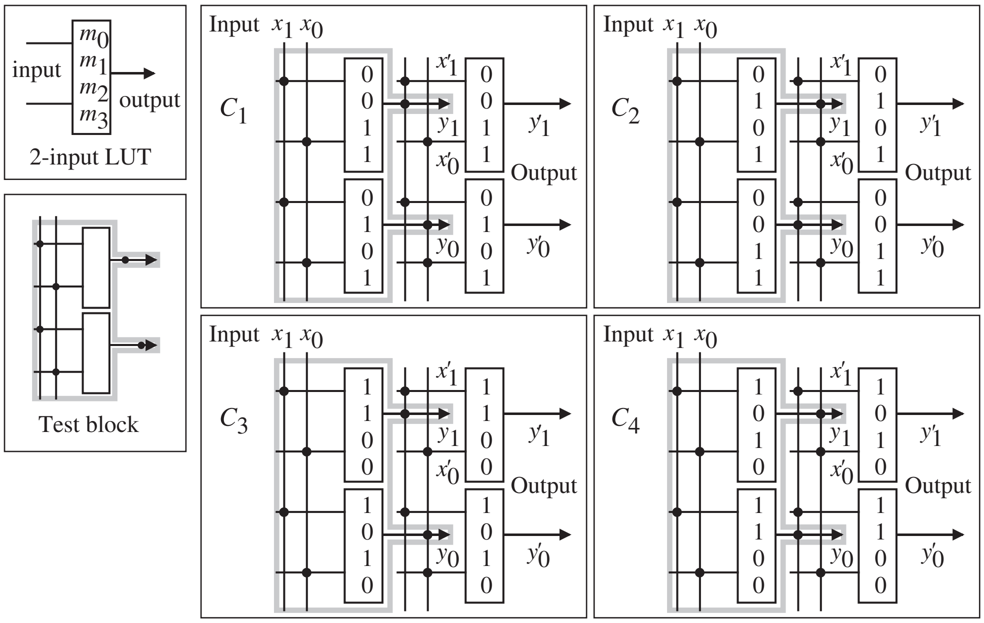
Here we consider universal test procedures for all LUTs in an FPGA. If we use any number of primary inputs/outputs for universal test, we can perform universal test procedure such as TP_1 and TP_2 for all LUTs in an FPGA concurrently. An FPGA, however, has a restricted number of I/O blocks, and hence input sequences cannot be applied to all the LUTs directly and simultaneously while the corresponding output values from all the LUTs are being observed. Therefore, for all LUTs in an FPGA, we have to consider universal test procedures under such a constraint of the limited number of I/O blocks.

First, we modify test procedure TP_1 into that for all LUTs. Suppose a set of k LUTs such that the i-th LUT implements a function $f_i(x) = b(x, k-i)$ for $1 \leq i \leq k$. If an input pattern a_j is applied to each of those LUTs concurrently, the output pattern obtained from the k LUTs is the same as the input pattern a_j, i.e., $(f_1(a_j), f_2(a_j), \ldots, f_k(a_j)) = a_j$. Note that the function $f_i(x)$ corresponds to the function that is implemented at the i-th configuration, C_i, in test procedure TP_1 for $1 \leq i \leq k$. Hence if the input sequences that are the same as those in test procedure TP_1 are applied those LUTs, then the k LUTs can be tested simultaneously while the resulting output sequences are applied to another set of k LUTs as input sequences, and consequently all LUTs in an FPGA can be tested.

Based on the above-mentioned method, we present a universal test procedure for all LUTs in an FPGA, called TP_1^{ALL}. Test procedure TP_1^{ALL} configures iterations of functional blocks such that each block consists of k LUTs and has k inputs and k outputs. Such a block is referred to as a *test block*. Let L_l denote the l-th LUT in each test block. The function implemented by LUT L_l at the i-th configuration in test procedure TP_1^{ALL} is the same as that by the LUT at the $(((i+l-2) \bmod k)+1)$ and $(k+((i+l-2) \bmod k)+1)$ in test procedure TP_1 for $1 \leq i \leq k$ and $k+1 \leq i \leq 2k$, respectively. In test procedure TP_1^{ALL}, all test blocks are connected with one another in a cascade. Test procedure TP_1^{ALL} applies the input sequence that is the same as that of test procedure TP_1 to the first test block in the cascade at each configuration. The output sequences from the first test block is applied to the second one. Similarly, the input sequences are applied to all the test block, and consequently all LUTs are tested simultaneously. Fig. 6 shows an example in case of $k=2$. Fig. 6(a) illustrates 4 ($=2k$) configurations for all LUTs in an FPGA. Here, two LUTs are combined into a single test block, shown outlined, and test blocks are cascaded from left to right. In this cascade, connections between test blocks are configured to generate the input/output sequences of Fig. 6(b). Note that the order of input sequences for even test blocks is the reverse of that for odd test blocks at configurations C_{k+1} through C_{2k}. However, all the input patterns $a_0, a_1, \ldots, a_{n-1}$ are applied to all LUTs at each configuration, and NAFs of which detection depends on the order of applied input sequences can be tested at the first k configurations. Therfore, in the same way as test procedure TP_1, we have the following theorem.

Theorem 3: Test procedure TP_1^{ALL} can detect any fault of all LUTs in an FPGA.

In the same way as TP_1^{ALL}, we present a test procedure called TP_2^{ALL} which is derived from test procedure TP_2. As shown in Fig. 5, in test procedure TP_2, only two input patterns that are different in the last bit (i.e., $\ldots 0$ and $\ldots 1$) are applied at each configuration. On the other hand, the output values for the two input patterns are 0 and 1. Hence, the output values 0 and 1 that are obtained from an LUT can be applied to the last input line of another LUT for all the configurations. That is, if we consider one LUT to be a test block, all LUTs can be tested simultaneously by connecting the output line of a test block (or an LUT) to the last input line of another test block. Fig. 7 shows an example of test procedure TP_2^{ALL} in case of $k=2$.



(a)

Config.	Input $x_1 x_0$	$y_1 y_0$	$x'_1 x'_0$	Output $y'_1 y'_0$
C_1	0 0	0 0	0 0	0 0
	0 1	0 1	0 1	0 1
	1 0	1 0	1 0	1 0
	1 1	1 1	1 1	1 1
	0 0	0 0	0 0	0 0
C_2	0 0	0 0	0 0	0 0
	0 1	1 0	0 1	1 0
	1 0	0 1	1 0	0 1
	1 1	1 1	1 1	1 1
C_3	0 0	1 1	1 1	0 0
	0 1	1 0	1 0	0 1
	1 0	0 1	0 1	1 0
	1 1	0 0	0 0	1 1
C_4	0 0	1 1	1 1	0 0
	0 1	0 1	1 0	1 0
	1 0	1 0	0 1	0 1
	1 1	0 0	0 0	1 1

(b)

Fig. 6. Example of test procedure TP_1^{ALL} ($k = 2$). (a) Configurations. (b) Input/output sequences.

Fig. 7(a) illustrates 4 ($= 2^k$) configurations for all LUTs in an FPGA. Here, one LUT is a single test block, shown outlined, and test blocks are cascaded from left to right. As shown in this figure, in test procedure TP_2^{ALL}, the function implemented by each LUT at the i-th configuration is the same as that in test procedure TP_2. The first and the third columns of Fig. 7(b) show the input sequences for the leftmost and the next test blocks, respectively. Note that when input sequence (a_j, a_l) (denoted at Lines 15 and 21 in Fig. 5) is applied to the first test block at odd configuration, the input sequence for even test blocks becomes (a_l, a_j). In order to feed the sequence (a_j, a_l) to even test blocks, test procedure TP_2^{ALL} applies the sequence (a_j, a_l, a_j) to the first test block at odd configurations. Consequently, the sequence (a_l, a_j, a_l) which includes (a_j, a_l) can be applied to even test blocks. Thus, in the same way as test procedure TP_2, we have the following theorem.

Theorem 4: Test procedure TP_2^{ALL} can detect any fault of all LUTs in an FPGA.

IV. Universal Test Complexity

The *universal test complexity* of a universal test procedure for an FPGA refers to the time required to perform the universal test procedure for the FPGA. In this section, we consider the universal test complexities of the universal test procedures mentioned above, TP_1^{ALL} and TP_2^{ALL} for SL-FPGAs and RAL-FPGAs.

Suppose a universal test procedure for an FPGA G such that

$$TP = \langle (C_1, S_1), (C_2, S_2), \ldots, (C_{n_c}, S_{n_c}) \rangle$$

where n_c is the number of configurations. Let $c(i)$ be the number of configuration memory cells that are loaded to implement the i-th configuration C_i. Let $s(i)$ be the length of input sequence S_i for the i-th configuration C_i. The time required to implement all the configurations in test procedure TP for FPGA G is given by

$$T_G^C(TP) = \sum_{i=1}^{n_c} t_c c(i)$$

where t_c is the time required to load one bit of a program into configuration memory cell in FPGA G. The time required to apply all the input sequences in test procedure TP for FPGA G is given by

$$T_G^S(TP) = \sum_{i=1}^{n_c} t_s s(i)$$

where t_s is the clock cycle time of a configuration implemented in FPGA G. Thus, the universal test complexity of test procedure TP for FPGA G is given by

$$\begin{aligned} T_G(TP) &= T_G^C(TP) + T_G^S(TP) \\ &= \sum_{i=1}^{n_c} (t_c c(i) + t_s s(i)). \end{aligned} \qquad (7)$$

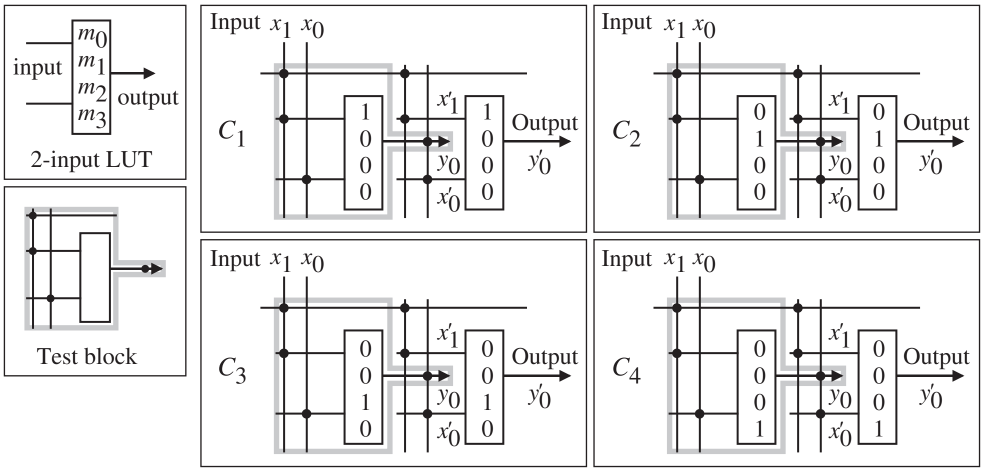
A. Test Complexity for SL-FPGA

Whenever a configuration is changed to another one on an FPGA, all the program bits to be loaded are required. Hence, the time required to implement a configuration of a test procedure TP for an SL-FPGA is expressed as

$$c(i) = N_m$$

for all i, where N_m is the total number of configuration memory cells in FPGA G. From Fig. 4, the number of configurations in test procedure TP_1^{ALL} is expressed as

$$n_c = 2k = 2 \log n$$



(a)

Config.	Input $x_1 x_0$	y_0	$x'_1 x'_0$	Output y'_0
C_1	0 0	1	0 1	0
	0 1	0	0 0	1
	0 0	1	0 1	0
C_2	0 1	1	0 1	1
	0 0	0	0 0	0
C_3	1 0	1	1 1	0
	1 1	0	1 0	1
	1 0	1	1 1	0
C_4	1 1	1	1 1	1
	1 0	0	1 0	0

(b)

Fig. 7. Example of test procedure TP_2^{ALL} ($k = 2$). (a) Configurations. (b) Input/output sequences.

where k is the number of input lines of an LUT and n is the size of an LUT, i.e., $n = 2^k$. Hence,

$$T_{SL}^{C}(TP_1^{ALL}) = 2t_c N_m \log n. \tag{8}$$

From Fig. 4, the length of input sequence for each configuration is expressed as

$$\begin{cases} s(1) = n+1 \\ s(i) = n \qquad \text{for} \quad 2 \le i \le n_c \end{cases}$$

Hence,

$$\begin{aligned} T_{SL}^{S}(TP_1^{ALL}) &= \sum_{i=1}^{n_c} t_s s(i) \\ &= t_s(2n \log n + 1). \end{aligned} \tag{9}$$

Note that Equation (9) holds independent of the programming scheme of the FPGA. Therefore, from Equations (8) and (9), the universal test complexity of test procedure TP_1^{ALL} for an SL-FPGA is expressed as

$$T_{SL}(TP_1^{ALL}) = t_c(2N_m \log n) + t_s(2n \log n + 1) \tag{10}$$

Without loss of generality, the total number of configuration memory cells in an FPGA is assumed to be proportional to the array size of the FPGA (or the number of CLBs) and the size of an LUT (or the number of memory cells in an LUT), i.e.,

$$N_m = O(Nn)$$

where N is the array size of the FPGA or the number of LUTs in the FPGA. Hence,

$$T_{SL}(TP_1^{ALL}) = O(Nn \log n). \tag{11}$$

In test procedure TP_2^{ALL}, from Figs. 5 and 7, the number of configurations is $n_c = n$, and the length of input sequence for the i-th configuration is

$$s(i) = \begin{cases} 3 & \text{if } i \text{ is odd} \\ 2 & \text{otherwise} \end{cases}$$

Hence, we have

$$T_{SL}^{C}(TP_2^{ALL}) = N_m n t_c \tag{12}$$

and

$$T_{SL}^{S}(TP_2^{ALL}) = \frac{5}{2} n t_s. \tag{13}$$

Note that Equation (13) also holds independent of the programming scheme of the FPGA as well as Equation (9). Therefore, from Equations (12) and (13), the universal test complexity of test procedure TP_2^{ALL} for an SL-FPGA is expressed as

$$\begin{aligned} T_{SL}(TP_2^{ALL}) &= N_m n t_c + \frac{5}{2} n t_s \\ &= O(Nn^2) \end{aligned} \tag{14}$$

From Equations (10) and (14), we can see that for SL-FPGAs the universal test complexity of test procedure TP_1^{ALL} is smaller than that of test procedure TP_2^{ALL}, and hence test procedure TP_1^{ALL} is more appropriate than test procedure TP_2^{ALL} for SL-FPGAs. Thus, we have the following theorem.

Theorem 5: There exists a universal test procedure for SL-FPGAs such that the universal test complexity is $O(Nn \log n)$.

B. Test Complexity for RAL-FPGA

The number of bits loaded to implement the first configuration of a test procedure for an RAL-FPGA is expressed as

$$c(1) = N_m. \tag{15}$$

Note that Equation (15) holds independent of the test procedure. In test procedure TP_1^{ALL}, the number of configuration memory cells that are different from those of the previous configuration is expressed as $n/2$ for each LUT. The connection of the output line of each LUT is changed at every configuration to feed the output sequences from a test block to another one. Hence,

$$c(i) = N(\frac{n}{2} + 2) \tag{16}$$

for $2 \leq i \leq n_c$. Accordingly,

$$T^C_{RAL}(TP^{ALL}_1) = t_c(N_m + \sum_{i=2}^{2\log n} N(\frac{n}{2}+2))$$
$$= t_c(N_m + N((n+4)\log n - \frac{n}{2} - 2). \quad (17)$$

From Equations (17) and (9), we have

$$T_{RAL}(TP^{ALL}_1) = O(Nn\log n). \quad (18)$$

In test procedure TP^{ALL}_2, only two configuration memory cells in an LUT are changed at each configuration. Hence,

$$\begin{aligned} T^C_{RAL}(TP^{ALL}_2) &= t_c(N_m + \sum_{i=2}^{n} 2N) \\ &= t_c(N_m + 2N(n-1)). \quad (19) \end{aligned}$$

From Equations (19) and (13), we have

$$\begin{aligned} T_{RAL}(TP^{ALL}_2) &= t_c(N_m + 2N(n-1)) + 2nt_s \\ &= O(Nn). \quad (20) \end{aligned}$$

From Equations (18) and (20), we can see that for RAL-FPGAs the universal test complexity of test procedure TP^{ALL}_2 is smaller than that of test procedure TP^{ALL}_1, and hence test procedure TP^{ALL}_2 is more appropriate than test procedure TP^{ALL}_1 for RAL-FPGAs. Thus, we have the following theorem.

Theorem 6: There exists a universal test procedure for RAL-FPGAs such that the universal test complexity is $O(Nn)$.
From Theorems 5 and 6, we can consider that RAL-FPGAs are more easily-testable than SL-FPGAs.

C. C-testable FPGA

Since an FPGA consists of an array of logic blocks, it can be considered to be one of iterative systems. C-testable [6] is a term which expresses an important class of testable iterative systems. Here we consider a C-testable FPGA.

Definition 1 (C-testable) An FPGA is said to be *C-testable* if there exists a universal test procedure such that the universal test complexity for the FPGA is independent of the array size.

As shown by Theorems 5 and 6, either SL-FPGAs or RAL-FPGAs may not be C-testable. In each of test procedures TP^{ALL}_1 and TP^{ALL}_2, however, if we regard each test block as a logic cell, each configuration can be considered to be an iterative system. Note that the length of input sequences applied to each configuration is independent of the array size of FPGAs (Equations (9) and (13)). On the other hand, from Equations (8) and (19), we can see that the time required to load programs for testing FPGAs depends on the array size of the FPGAs. Therefore, if an FPGA can load the same program into each test block simultaneously, the time required to load programs will be independent of the array size. Such type of programming schemes is called *block-sliced loading*, and SL-FPGAs and RAL-FPGAs with block-sliced loading are referred to as *BSSL-FPGAs* and *BSRAL-FPGAs*, respectively.

Let t_b be the time required to load the same bit into the corresponding configuration memory cell of each test block. Then, we have the following theorems.

Theorem 7: BSSL-FPGAs are C-testable.

Proof: In test procedure TP^{ALL}_1, the number of LUTs in a test block is $k = \log n$. The number of configuration memory cells of an LUT and the number of input and output lines of each LUT are $n = 2^k$ and $k+1 = \log n + 1$, respectively (See Fig. 6). Hence,

$$T^C_{BSSL}(TP^{ALL}_1) = \sum_{i=1}^{2\log n} t_b \log n(n + \log n + 1). \quad (21)$$

From Equations (21) and (9), we have

$$T_{BSSL}(TP^{ALL}_1) = O(n\log^2 n + \log^3 n). \quad (22)$$

■

Theorem 8: BSRAL-FPGAs are C-testable.

Proof: The test block in test procedure TP^{ALL}_2 consists of only one LUT. The number of configuration memory cells of an LUT and the number of input and output lines of each LUT are $n = 2^k$ and $k+1 = \log n + 1$, respectively. $(k-1)$ lines from I/O blocks are connected to each test block. At each configuration, only two configuration memory cells in an LUT are changed (See Fig. 7). Hence,

$$c(i) = \begin{cases} n + 2\log n & \text{for } i = 1 \\ 2 & \text{for } 2 \leq i \leq n \end{cases} \quad (23)$$

Therefore,

$$T^C_{BSRAL}(TP^{ALL}_2) = t_b(3n + 2\log n - 2). \quad (24)$$

From Equations (24) and (13), we have

$$T_{BSRAL}(TP^{ALL}_2) = O(n + \log n). \quad (25)$$

■

Thus, we can have C-testable FPGAs by the programming scheme of block-sliced loading.

V. Conclusions

In this paper, we considered *universal test* such that when applied to an *un*programmed FPGA, it ensures that *all* the corresponding programmed logic circuits on the FPGA are fault-free. We presented two types of programming schemes; *sequential loading* and *random access loading*, and showed test procedures for the FPGAs with these programming schemes and their *test complexities*. In order to make the test complexity for FPGAs independent of the array size of the FPGAs, we proposed a programming scheme called *block-sliced loading*, which makes FPGAs *C-testable*.

In this paper, we focused on testing for look-up tables in FPGAs. However, testing for other components, e.g. I/O blocks and interconnect structures, are also important. These components can be tested in the same way as testing for look-up tables. We will report in the near future on the testing for these components as well as the whole of FPGAs.

Acknowledgments

The authors would like to thank Dr. Toshimitsu Masuzawa and Dr. Michiko Inoue of Nara Institute of Science and Technology for their helpful comments and discussions on this work.

References

[1] S.D. Brown, R.J. Francis, J. Rose, and S.G. Vranesic, *Field-Programmable Gate Arrays*, Kluwer Academic Publishers, 1992.

[2] S.M. Trimberger, Ed., *Field-Programmable Gate Array Technology*, Kluwer Academic Publishers, 1994.

[3] *The Programmable Logic Data Book*, Xilinx, 1994.

[4] M. Hermann and W. Hoffmann, “Fault modeling and test generation for FPGAs,” in *Lecture Notes in Computer Science, Field Programmable Logic*, R.W. Hartenstein and M.Z. Servít, Eds. 1994, pp. 1–10, Springer-Verlag.

[5] R.O. Durate and M. Nicolaidis, “A test methodology applied to cellular logic programmable gate arrays,” in *Lecture Notes in Computer Science, Field Programmable Logic*, R.W. Hartenstein and M.Z. Servít, Eds. 1994, pp. 11–22, Springer-Verlag.

[6] A.D. Friendman, “Easily testable iterative systems,” *IEEE Trans. Comput.*, vol. C-22, no. 12, pp. 1061–1064, Dec. 1973.

[7] M.S. Abadir and H.K. Reghbati, “Functional testing of semiconductor random access memories,” *ACM Computing Surveys*, vol. 15, no. 3, pp. 175–198, Mar. 1983.

[8] A. Tuszynski, “Memory testing,” in *VLSI Testing*, T.W. Williams, Ed., pp. 161–228. Elsevier Science Publishers, 1986.

[9] Tomoo Inoue, Hideo Fujiwara, Hiroyuki Michinishi, Tokumi Yokohira, and Takuji Okamoto, “Universal test complexity of field-programmable gate arrays,” in *the Forth IEEE Asian Test Symp.*, Nov. 1995, pp. 259–265.