FPGA Interconnect Delay Fault Testing

Erik Chmelař
Center for Reliable Computing
Stanford University

Abstract

The interconnection network consumes the majority of die area in an FPGA. Presented is a scalable manufacturing test method for all SRAM-based FPGAs, able to detect multiple interconnect delay faults, multiple bridging faults, or both. An adjustable maximum sensitivity to resistive open defects of several kilo-ohms is achieved. A bridging fault that causes a signal transition to occur on at least one of the bridged interconnects is detectable. Finally, fast and simple fault location is presented.

1 Introduction

A satisfactory FPGA testing method meets several criteria. First, the routing resources must be explicitly tested. Approximately 80% of the transistors in an FPGA are dedicated to the routing resources; thus, targeting interconnect faults is necessary. Second, a high fault coverage must be achieved. The testing method should detect the major interconnect fault types, specifically stuck-at, stuck open, delay, and bridging faults (stuck-at and stuck open faults can each be modeled as a delay fault of infinite delay). Third, latent defects (defects that are typically not detectable during manufacturing testing and result in *early life failures*) should be detected. Screening out these early life failures improves device reliability. Finally, the testing method should have the ability to locate a faulty resource. Fault location is important for device yields in two ways: it is an important step in fault diagnosis, which is performed to collect information necessary for manufacturing process improvements, and it is crucial to the *application-dependent FPGA* model, in which a customer's design functions correctly on a faulty device by not using the faulty resource or resources.

The organization of this paper is as follows. A generic FPGA structure is given in Sec. 2. Recent work in FPGA delay fault testing is surveyed in Sec. 3. The proposed delay fault testing method is presented in Sec. 4: a high-level overview is provided in Sec. 4.1, the necessary FPGA configurations are discussed in Sec. 4.2, and the threshold timing parameters are defined in Sec. 4.3. Next, the faults detected by the presented method are discussed in Sec. 5: resistive open defects are addressed in Sec. 5.1 and bridging faults are addressed in Sec. 5.2. Fault location is presented in Sec. 6. Finally, the paper concludes in Sec. 7.

2 FPGA Structure

The principal building blocks of an FPGA are *logic blocks*, *switch matrices* or *programmable multiplexers* or both, and *input/output blocks* (special multiplier blocks and block RAMs are not addressed in this paper). A logic block contains the combinational and sequential elements needed to perform logic functions: *SRAM Look-up Tables* (LUTs) implement combinational functions, and bistable elements (configurable as either latches or flip-flops) are used in sequential designs. A switch matrix (or programmable multiplexer) is used to join wire segments to form physical paths between blocks. Wire segments incident to a switch matrix are joined via one or more *Programmable Interconnect Points* (PIPs), a pass transistor controlled by a memory cell that determines whether the PIP is *on* (conducting) or *off* (non-conducting). Finally, an input/output block is used to pass signals between an FPGA and an external device.

The logic blocks are placed regularly throughout an FPGA, with the input/output blocks along the periphery. The highly regular layout simplifies the routing of wires, grouping those of similar lengths (and similar beginning and ending points) together into a common bus.

3 Previous Work

There are effective well-known methods for testing FPGA logic blocks [1] [2] [3]. Additionally, various techniques for testing FPGA routing resources have been proposed [4] [5] [6] [7] [8] [9], addressing stuck-at, stuck open, and bridging faults. Recently, explicit testing for delay faults in FPGAs has been addressed [10] [11] [12] [13], as it is suspected that 58% of customer-returned semiconductor chips have open defects [14]. An open defect can be a partial open, in which the circuit connection is present but not completely conductive, or it can be a complete open, in which no circuit connection is present.

The 'added fanout' method [10], targeting resistive open defects in the routing resources, consists of adding an additional load (fanout) to a signal *Path Under Test* (PUT) by turning on some predetermined number of PIPs that connect dummy paths to the PUT. The added load increases the delay of the PUT, causing it to be greater than the clock cycle if a resistive open defect above some threshold is present.

The 'oscillator-loop' method [11], targeting delay faults in both the routing and logic resources, consists of configuring an FPGA into several *Iterative Logic Arrays* (ILAs) of PUTs. A signal transition is simultaneously driven at the input of each of the ILAs, and the difference in the propagation delays is measured at the outputs of the ILAs. An on-chip oscillator pulses for the duration between when the first signal transition occurs at the output to when the last signal transition occurs. A delay fault is detected when the oscillator pulse count is greater than some predetermined threshold value.

4 Delay Fault Testing

4.1 Overview

The presented manufacturing testing method, also suitable for fault diagnosis, exploits the bus-oriented wiring structure of FPGAs. The basic idea is to test a set of interconnects, or paths, between two logic blocks for delay faults by creating a race condition between the signals propagating on those paths. The particular set of paths under test (PUTs) between two logic blocks are configured such that their fault-free propagation delays are nearly identical, and thus a signal transition simultaneously occurring at the start of the PUTs should also simultaneously occur the end of the PUTs. The bus-oriented wiring structure easily facilitates the configuration of such equal-delay paths. Any signal transition that occurs at the destination logic block substantially later than the first occurring signal transition signifies the presence of a delay fault. Note that terms like *equal*, *identical*, *simultaneous*, and *at the same time* are not used in the pedantically correct manner, since it is impossible to have a set of PUTs with *exactly* the same propagation delays. An analysis of tolerable propagation delay differences is provided in the discussion pertaining to Table 2 in Sec. 4.3.

The FPGA under test is configured to have one or more independent iterative logic arrays (ILAs), on each of which propagates a single signal transition. An ILA is a series of M logic blocks, $LB_0 \ldots LB_{M-1}$, and the associated interconnects (PUTs). Each logic block, LB_i, is connected to a preceding logic block, LB_{i-1} and a succeeding logic block, LB_{i+1}. The connections between two logic blocks, say LB_{i-1} and LB_i, form a set, set_i, of $2N$ PUTs*, $PUT_{i_1} \ldots PUT_{i_{2N}}$, whose differences in propagation delays from LB_{i-1} to LB_i are negligible. A portion of an ILA is shown in Fig. 1.

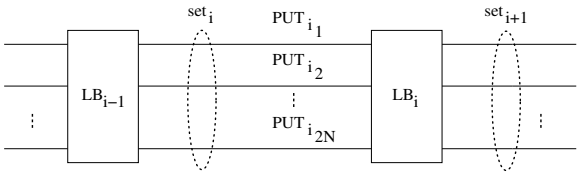


Figure 1: Iterative Logic Array

The testing procedure begins at the first logic block of an ILA, LB_0, which simultaneously drives a signal transition on each of the $2N$ PUTs of set_1. When no PUT of set_1 has a delay fault, the propagation delays to LB_1 are all identical, and thus the signal transition on each PUT occurs at LB_1 at the same time. However, if up to $2N-1$ of the PUTs have a delay fault, a race condition is set up between the PUT with the smallest propagation delay (the fastest PUT) and the PUT with the largest propagation delay (the slowest PUT). A delay fault is detected if the phase difference between the fastest and slowest PUTs is above some threshold value.

The logic blocks of a particular ILA are configured identically (with the exception of the first logic block, LB_0). Each logic block, LB_i, serves as a destination for the signals of set_i and as a source for the signals of set_{i+1} Thus, the logic implemented within each logic block performs two functions: (1) to detect whether a delay fault is present on any of the preceding PUTs, and (2) to propagate a signal transition on the succeeding PUTs.

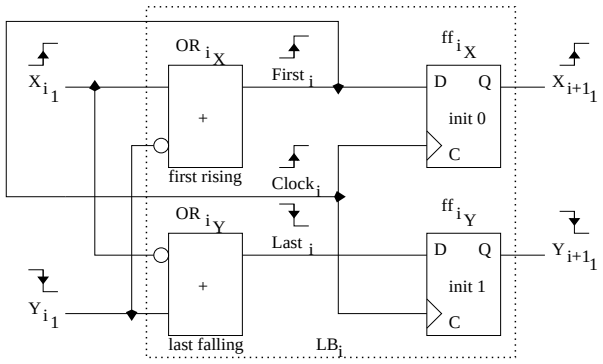
4.2 FPGA Configuration

4.2.1 Signal Transitions

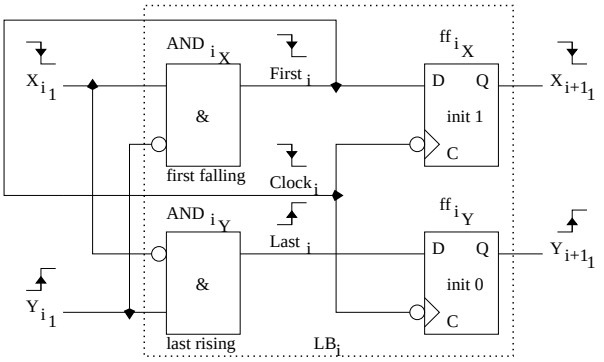
To detect both rising and falling delay faults on each PUT, two testing phases are needed, one to drive a 0 to 1 transition on a particular PUT (test phase A), and a second to drive a 1 to 0 transition on that same PUT (test phase B). The two test phases, A and B, correspond to two distinct logic block configurations: test phase A is shown as configuration *A* in Fig. 2a and test phase B is shown as configuration *B* in Fig. 2b. The two logic block configurations are similar: configuration *B* is simply derived by applying DeMorgan's Law to configuration *A*, swapping OR with AND, 0 with 1, X with $\overline{X}$, and Y with $\overline{Y}$. Thus, all statements about test phase A apply equally to test phase B, *mutatis mutandis*, and therefore, unless otherwise stated, only test phase A (configuration *A*) is discussed in detail in this paper.

The PUTs of a set, say set_i, are partitioned into two groups, the $\mathbf{X}_i = X_{i_1} \ldots X_{i_N}$ group and the $\mathbf{Y}_i = Y_{i_1} \ldots Y_{i_N}$

*It is not necessary to have an even number of PUTs: an even number is chosen, without loss of generality, to simplify the analysis.



(a) Configuration *A*: LB_i



(b) Configuration *B*: LB_i

Figure 2: Logic Block Configurations

group[†]. Figure 2 shows the simple case where set_i consists of only two PUTs ($2N = 2$), partitioned into the necessary two groups, $\mathbf{X}_i = X_{i_1}$ and $\mathbf{Y}_i = Y_{i_1}$.

Each of the X_i PUTs always carries an identical signal, opposite in polarity to each of the Y_i PUTs (each of the Y_i PUTs also always carries an identical signal). In test phase A, PUTs of the $\mathbf{X}_i$ group are each initialized to 0 and each incurs (in the fault-free case) a single signal transition from 0 to 1. The signal propagation time of a PUT is Δt, thus a signal transition from 0 to 1 on each of the N X_i PUTs at time t, occurring within a time period Δt is written

$$\begin{aligned}
&\{X_{i_1}(t)\ldots X_{i_N}(t)\} \rightarrow \{X_{i_1}(t+\Delta t)\ldots X_{i_N}(t+\Delta t)\} \\
&= \{\mathbf{X}_i(t)\} \rightarrow \{\mathbf{X}_i(t+\Delta t)\} \\
&= \{0\ldots 0\}_{\mathbf{X}_i} \rightarrow \{1\ldots 1\}_{\mathbf{X}_i} \\
&= \{\mathbf{0}\}_{\mathbf{X}_i} \rightarrow \{\mathbf{1}\}_{\mathbf{X}_i}.
\end{aligned}$$

Similarly, all PUTs of the $\mathbf{Y}_i$ group are each initialized to 1 and each incurs (in the fault-free case) a single signal transition from 1 to 0, written $\{\mathbf{1}\}_{\mathbf{Y}_i} \rightarrow \{\mathbf{0}\}_{\mathbf{Y}_i}$.

[†]Any partitioning in which there is at least one PUT in the $\mathbf{X}_i$ group and one in the $\mathbf{Y}_i$ group is valid.

A delay fault is detected when one or more signals do not transition within the Δt time period. For example, a set_i, with 2 X_i PUTs and 2 Y_i PUTs, is fault free if all signals transition, written

$$\{00,11\}_{\mathbf{X}_i,\mathbf{Y}_i} \rightarrow \{11,00\}_{\mathbf{X}_i,\mathbf{Y}_i}$$

while the set_i fails if, say, the second X_i PUT does not transition, written

$$\{00,11\}_{\mathbf{X}_i,\mathbf{Y}_i} \rightarrow \{10,00\}_{\mathbf{X}_i,\mathbf{Y}_i}.$$

In general, a single signal transition occurring within a time period of Δt on each of the X_i and Y_i PUTs, driven simultaneously by LB_{i-1}, is called a *Pass Signal Transition* (PST), written (for test phase A)

$$PST_i = \{\mathbf{0},\mathbf{1}\}_{\mathbf{X}_i,\mathbf{Y}_i} \rightarrow \{\mathbf{1},\mathbf{0}\}_{\mathbf{X}_i,\mathbf{Y}_i}.$$

Additionally, a single signal transition occurring within a time period of Δt on each of the X_i PUTs only (all of the signals of the Y_i PUTs remain constant), driven simultaneously by LB_{i-1}, is called a *Fail Signal Transition* (FST), written (for test phase A)

$$FST_i = \{\mathbf{0},\mathbf{1}\}_{\mathbf{X}_i,\mathbf{Y}_i} \rightarrow \{\mathbf{1},\mathbf{1}\}_{\mathbf{X}_i,\mathbf{Y}_i}.$$

Note that the signal transitions corresponding to a failing set_i (arbitrary transitions) and those corresponding to an FST (all X_i PUTs transition, all Y_i PUTs remain constant) are not the same: an FST corresponds to the signal transitions that are propagated down an ILA after a delay fault is detected (discussed in Sec. 4.2.2).

Figures 4a and 4b display an FST and a PST, respectively, for the simple case where $2N = 2$.

4.2.2 Logic Block Configurations

In test phase A, within a particular logic block, say LB_i, one look-up table (LUT) is configured to generate a 0 to 1 transition on its output when the first signal transition occurs on any of the PUTs of set_i. This LUT, labeled OR_{i_X} in Fig. 2a, generates the $First_i$ signal, implementing $First_i = X_{i_1} + \ldots + X_{i_N} + \overline{Y_{i_1}} + \ldots + \overline{Y_{i_N}} = \mathbf{X}_i + \overline{\mathbf{Y}_i}$. $First_i$ is input to a D flip-flop, ff_{i_X}. Similarly, another LUT of LB_i is configured to generate a 1 to 0 transition on its output only after a signal transition has occurred on *every* PUT of set_i. This LUT, labeled OR_{i_Y} in Fig. 2a, generates the $Last_i$ signal, implementing $Last_i = \overline{X_{i_1}} + \ldots + \overline{X_{i_N}} + Y_{i_1} + \ldots + Y_{i_N} = \overline{\mathbf{X}_i} + \mathbf{Y}_i$. $Last_i$ is input to a D flip-flop, ff_{i_Y}. Note that if there is more than one PUT in a single group ($\mathbf{X}_i$ or $\mathbf{Y}_i$), the fanout from the single flip-flop output (ff_{i_X} or ff_{i_Y}, respectively) to multiple PUTs is accomplished by the routing resources, for example by a switch matrix.

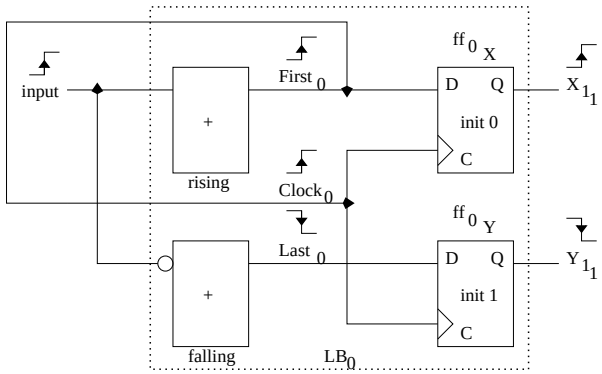
The output of the LUT generating the $First_i$ signal (OR_{i_X} in configuration *A*) is fed back to the clock input of ff_{i_X} and ff_{i_Y}, clocking each flip-flop by a 0 to 1 signal transition on $Clock_i$. $Clock_i$ is a time-delayed version of $First_i$ due to the feedback path's propagation delay. If the signals of all PUTs of set_i correctly transition at the same time, the $Last_i$ signal is generated by OR_{i_Y}, transitioning from 1 to 0, at the same time that the $First_i$ signal is generated by OR_{i_X}, and thus the subsequent clocking of the flip-flops by $Clock_i$ will capture the correct values of $First_i$ and $Last_i$, 1 and 0, respectively. In this case a PST propagates through LB_i, since LB_i drives a $\{\mathbf{0},\mathbf{1}\}_{\mathbf{X}_{i+1},\mathbf{Y}_{i+1}} \rightarrow \{\mathbf{1},\mathbf{0}\}_{\mathbf{X}_{i+1},\mathbf{Y}_{i+1}}$ transition on set_{i+1}. When the first signal transition of set_{i+1} occurs at the succeeding logic block, LB_{i+1}, it performs the same function as did LB_i. LB_{i+1} subsequently propagates a PST if it does not detect a delay fault on any of the $2N$ PUTs of set_{i+1}. In this manner the PST propagates down the ILA, causing each logic block to determine whether up to $2N-1$ delay faults are present on the preceding set of $2N$ PUTs.

If, however, not all the signals of the PUTs of set_i correctly transition in time to generate the $Last_i$ signal before the flip-flops are clocked, the $Last_i$ signal will not have made the correct 1 to 0 transition, and thus ff_{i_Y} will capture a value of 1, instead of 0, when it is clocked. Thus, an FST is propagated by LB_i, since LB_i drives a $\{\mathbf{0},\mathbf{1}\}_{\mathbf{X}_{i+1},\mathbf{Y}_{i+1}} \rightarrow \{\mathbf{1},\mathbf{1}\}_{\mathbf{X}_{i+1},\mathbf{Y}_{i+1}}$ transition on set_{i+1} (the signal on each Y_{i+1} PUT does not transition). In this case, the succeeding logic block, LB_{i+1}, sees a $\{\mathbf{0},\mathbf{1}\}_{\mathbf{X}_{i+1},\mathbf{Y}_{i+1}} \rightarrow \{\mathbf{1},\mathbf{1}\}_{\mathbf{X}_{i+1},\mathbf{Y}_{i+1}}$ transition on its inputs. Consequently, the $First_{i+1}$ signal is generated after the first signal transition on $\mathbf{X}_{i+1}$ occurs at LB_{i+1}, but the $Last_{i+1}$ signal remains constant at 1, since no signal on any Y_{i+1} PUT transitions. Thus, when the $Clock_{i+1}$ signal transitions, the output of ff_{i+1_X} transitions from 0 to 1 but the output of ff_{i+1_Y} remain constant. In this manner the propagation of the FST continues to the end of the ILA.

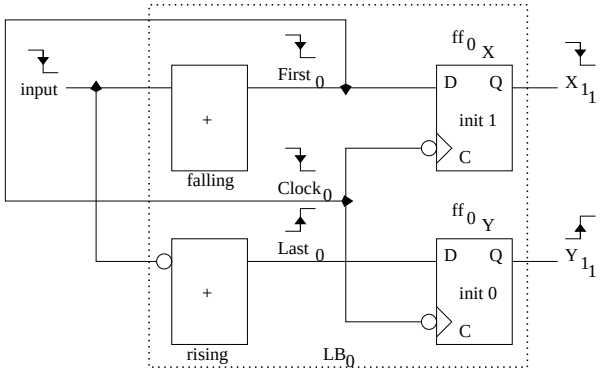
4.2.3 Starter and Compactor Blocks

In addition to the logic block configurations described for test phases A and B, a *starter block*, LB_0, must be placed at the head of an ILA to begin the PST propagation. The starter block is simply a logic block configured to have a single input (instead of multiple inputs) that causes the outputs of the flip-flops, ff_{0_X} and ff_{0_Y}, to transition at the same time. The outputs of LB_0 are the inputs to LB_1. The presence of a starter block eliminates any skew that may be encountered if multiple signals are brought from off-chip directly into LB_1, which would undoubtedly cause LB_1 to erroneously detect a delay fault on set_1. Figures 3a and 3b show the starter block configurations for test phases A and B, respectively.

The output of an *M*-length ILA (LB_0 through LB_{M-1}) is comprised of two signals (outputs of ff_{M-1_X} and ff_{M-1_Y}). A *compactor block* (not shown) can be used



(a) Configuration *A*: LB_0



(b) Configuration *B*: LB_0

Figure 3: Starter Block Configurations

to reduce the two signals into one. The compactor block for test phase A is simply a configuration of a single LUT inside the compactor block to implement the AND of the expected values stored in ff_{M-1_X} and ff_{M-1_Y}, $result = X_{M-1} \cdot \overline{Y_{M-1}}$, mapping $\{1,0\}_{X_{M-1},Y_{M-1}}$ to $\{1\}_{X_{M-1}}$ (pass) and $\{1,1\}_{X_{M-1},Y_{M-1}}$ to $\{0\}_{X_{M-1}}$ (fail). The function implemented by the LUT of the compactor block for test phase B is $result = \overline{X_{M-1}} \cdot Y_{M-1}$, mapping $\{0,1\}_{X_{M-1},Y_{M-1}}$ to $\{1\}_{X_{M-1}}$ (pass) and $\{0,0\}_{X_{M-1},Y_{M-1}}$ to $\{0\}_{X_{M-1}}$ (fail).

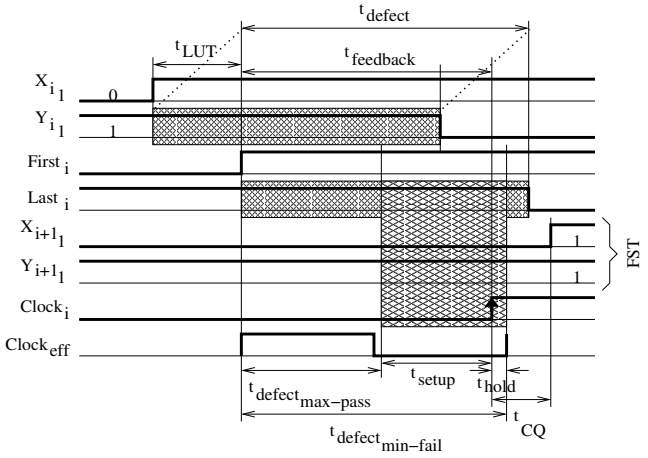
4.2.4 ILA Scalability

Note that within a testing phase (A or B), the configurations of the logic blocks ($LB_1 \ldots LB_{M-1}$) are all identical. Thus, the creation of ILAs is a scalable procedure, independent of FPGA array size (total number of logic blocks), requiring only the instantiation of identical logic blocks throughout the FPGA,

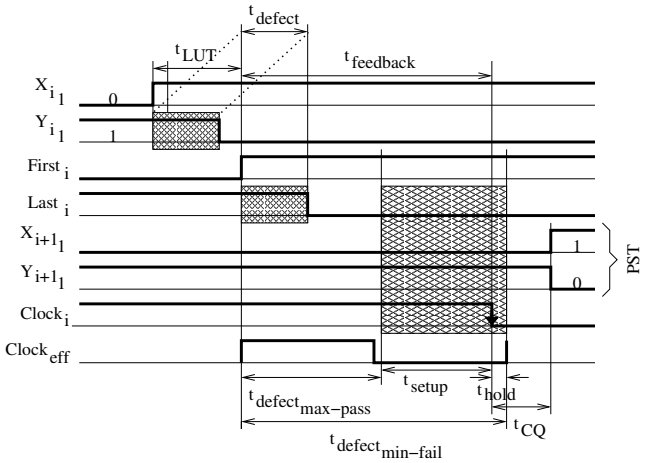
The connections between logic blocks is easily accomplished automatically by place-and-route software tools.

Furthermore, since only the functionality of the logic blocks changes between testing phases, partial reconfiguration can be used. This dramatically speeds up the total

test time, since the dominant portion of FPGA test time is consumed by configuring the device [5].



(a) Above-threshold Delay in set_i (fail)



(b) Below-threshold Delay in set_i (pass)

Figure 4: Configuration *A* Waveforms

4.3 Timing Thresholds

A delay fault is detected on set_i when the $Last_i$ signal is not generated by OR_{i_Y} before the flip-flops of LB_i are clocked by $Clock_i$. To guarantee that a delay fault is detected (gross delay fault) the minimum defect delay that PUT_{i_j} can have is

$$t_{defect_{min-fail}} = t_{feedback} + t_{hold}, \quad (1)$$

where t_{hold} is the flip-flop hold time, and $t_{feedback}$ is the propagation delay of the feedback path (the delay through the LUT, t_{LUT}, is common to both the $First_i$ and $Last_i$ signals, and is therefore not included in Equations 1 and 2). The signal transition on $Clock_i$ occurs $t_{feedback}$ after the $First_i$ signal transitions; therefore, if the $Last_i$ signal does not transition until t_{hold} after the signal transition on $Clock_i$, ff_{i_Y} will capture the incorrect value when clocked. The defect delay on set_i causes LB_i to propagate an FST ($\{\mathbf{0},\mathbf{1}\}_{\mathbf{X}_{i+1},\mathbf{Y}_{i+1}} \rightarrow \{\mathbf{1},\mathbf{1}\}_{\mathbf{X}_{i+1},\mathbf{Y}_{i+1}}$ for test phase A) on set_{i+1} a clock-to-Q time period, t_{CQ}, after the clocking edge of $Clock_i$, shown in Fig. 4a.

Similarly, the maximum defect delay that PUT_{i_j} can have and still guarantee that set_i passes is

$$t_{defect_{max-pass}} = t_{feedback} - t_{setup}, \quad (2)$$

where t_{setup} is the flip-flop setup time, and $t_{feedback}$ is the propagation delay of the feedback path. The signal transition on $Clock_i$ occurs $t_{feedback}$ after the $First_i$ signal transitions; therefore, the $Last_i$ signal must transition t_{setup} prior to the signal transition on $Clock_i$ to guarantee that set_i passes. The absence of a delay fault on set_i causes LB_i to propagate a PST ($\{\mathbf{0},\mathbf{1}\}_{\mathbf{X}_{i+1},\mathbf{Y}_{i+1}} \rightarrow \{\mathbf{1},\mathbf{0}\}_{\mathbf{X}_{i+1},\mathbf{Y}_{i+1}}$ for test phase A) on set_{i+1} t_{CQ} after the clocking edge of $Clock_i$, shown in Fig. 4b.

A marginal delay fault on set_i, greater than $t_{defect_{max-pass}}$ and less than $t_{defect_{min-fail}}$, places the transition of the $Last_i$ signal at the D-input of ff_{i_Y} within its sampling window. In this case the output of ff_{i_Y} is unpredictable due to the timing violation, but will be interpreted by the succeeding logic block in one of three ways: (1) as a PST (if correct signals transitions occur), (2) as an FST (if no signal transitions occur), or (3) as a failing set_{i+1} (if arbitrary signal transitions occur). Table 1 shows the values stored in ff_{i_X} and ff_{i_Y} for no delay fault, one or more marginal (below-threshold) delay faults, and one or more gross (above-threshold) delay faults on set_j for test phases A and B, respectively.

Table 1: Flip-flop Contents for a Delay Fault in set_j

Cfg.	Delay Fault	Test Result	Post-test Flip-flop State $\{Q_{i_X}, Q_{i_Y}\}$		
			$i < j$	$i = j$	$i > j$
A: init {0,1}	none	pass	1,0	1,0	1,0
	marginal	pass	1,0	1,0	1,0
	marginal	fail	1,0	1,1	1,1
	gross	fail	1,0	1,1	1,1
B: init {1,0}	none	pass	0,1	0,1	0,1
	marginal	pass	0,1	0,1	0,1
	marginal	fail	0,1	0,0	0,0
	gross	fail	0,1	0,0	0,0

The value of $t_{defect_{max-pass}}$ can be considered an effective clock period of the sequential circuit implemented within each logic block ($Clock_{eff}$ is shown in Fig. 4): the circuit behaves as though the signal transition on $First_i$ is the first clocking edge of an imaginary clock signal input to the flip-flops and the signal transition on $Clock_i$ is the second clocking edge. The signal transition on $Last_i$ must occur

within the effective clock period of the imaginary clock signal, $Clock_{eff}$, without causing a setup-time violation.

The feedback path's propagation delay (delay from $First_i$ to $Clock_i$) can be configured to obtain an arbitrary effective clock period, and thus an arbitrary delay fault detection sensitivity. The length of the feedback path determines $t_{feedback}$, whose value is easily found using FPGA *Computer-aided Design* (CAD) tools. The value of $t_{feedback}$ can be further increased, if necessary, by adding transparent logic elements in the feedback path. Table 2 shows the relevant *minimum* timing parameters for several Xilinx device families. All values are calculated using the average of the minimum attainable feedback propagation delays, $t_{fb} = t_{feedback_{avg-min}}$: the delay of the feedback path must be set greater than the minimum in most cases to ensure that good devices are not being erroneously failed (especially in the case where $t_{d_{m-p}} = t_{defect_{max-pass}} < 0$). Note that the presented method is independent of both device family and vendor: the relevant timing parameters are shown only for Xilinx devices due to our access to Xilinx software tools. In the event where the outputs of the two flip-flops in a given logic block are not in close proximity (for example if the outputs are on opposite sides of the logic block), an additional timing margin must be provided in the feedback path to absorb any propagation delay differences in the paths of a set, thus making the difference in propagation delays negligible.

Table 2: Threshold Timing Parameters

Device	Speed Grade	Timing Parameters		
		t_{fb} (ps)	$t_{d_{m-p}}$ (ps)	$t_{d_{m-f}}$ (ps)
Spartan-II	fast	840	40	840
	slow		40	840
Spartan-IIE	fast	360	-340	360
	slow		-440	360
Virtex	fast	1020	420	1020
	slow		220	1020
Virtex-II	fast	970	670	900
	slow		600	880
Virtex-E	fast	370	-90	370
	slow		-230	370
Virtex-IIPro	fast	790	780	880
	slow		780	910

$t_{fb} = t_{feedback_{avg-min}}$ $\quad t_{d_{m-p}} = t_{defect_{max-pass}}$ $\quad t_{d_{m-f}} = t_{defect_{min-fail}}$

5 Fault Detectability

5.1 Resistive Open Defects

Using an RC model to calculate delay, a resistive open defect (partial open) on a segment of a PUT (for example a wire segment) can be approximated by an exponential decay of the voltage on that segment,

$$V(t) = V_{DD}(1 - exp(\frac{t}{(R_{segment} + R_{defect})C_{segment}})). \quad (3)$$

Note that the presented method is independent of the delay model used: the RC model is chosen here for simplicity, and does not affect the threshold timing parameters reported in units of time in Table 2. The propagation delay of a PUT is defined to be the time period from when the voltage on a signal, driven by LB_i, reaches the $V_{DD}/2$ point to when the voltage of the signal, at LB_{i+1}, reaches the $V_{DD}/2$ point. In this manner, the propagation delay of a PUT is the sum of the propagation delays on each segment of the PUT. Therefore, subtracting the time needed for the voltage on a fault-free PUT to reach $V_{DD}/2$ at the input of a logic block from the time needed for the voltage on a faulty PUT to reach $V_{DD}/2$ at the input of a logic block yields t_{defect}, the added delay due to the resistive open defect,

$$t_{defect} = -R_{defect}C_{segment}ln(\frac{1}{2}). \quad (4)$$

Solving this expression for R_{defect} at the two boundary conditions, $t_{defect_{max-pass}}$ and $t_{defect_{min-fail}}$, yields approximate defect resistance detection thresholds,

$$R_{defect_{max-pass}} = \frac{-t_{defect_{max-pass}}}{C_{segment}ln(\frac{1}{2})}, \quad (5)$$

$$R_{defect_{min-fail}} = \frac{-t_{defect_{max-fail}}}{C_{segment}ln(\frac{1}{2})}. \quad (6)$$

Table 3 shows the threshold resistance values for the FPGA devices considered in Table 2. The value of $C_{segment}$ is taken to be 0.5 pF in all cases—an approximate capacitance for an intermediate-length wire segment (including all load capacitors).

5.2 Bridging Faults

In the bus-oriented wiring structure of an FPGA, wires are routed adjacently to each other for substantial distances. The PUTs of a set, being of the roughly the same length and destined to the same location, are inevitably routed on a common bus alongside each other. Bridging faults between PUTs must therefore be considered.

The presented method always drives each of the signals in the $\mathbf{X}_i$ group to the opposite polarity of the signals in the $\mathbf{Y}_i$ group. The X_i and Y_i paths should be routed in an interleaved fashion on the common bus, such that each X_{i_j} path runs adjacently to a Y_{i_k} path or paths, and each Y_{i_k} path runs adjacently to an X_{i_j} path or paths. Maximizing both the number of adjacent opposite-polarity signals and the dis-

Table 3: Threshold Defect Resistance Parameters

Device	Speed Grade	Threshold Values	
		$Clock_{eff}$ (GHz)	$R_{defect_{min-fail}}$ (kΩ)
Spartan-II	fast	1.19	2.4
	slow	1.19	2.4
Spartan-IIE	fast	2.78	1.0
	slow	2.78	1.0
Virtex	fast	0.98	2.9
	slow	0.98	2.9
Virtex-II	fast	1.11	2.6
	slow	1.14	2.5
Virtex-E	fast	2.70	1.1
	slow	2.70	1.1
Virtex-IIPro	fast	1.14	2.5
	slow	1.10	2.6

All calculations use $C_{segment} = 0.5$ pF.

tance over which the paths carrying such signals run alongside each other maximizes the probability that a bridging fault between PUTs will be detected. The interleaving of a set_i, consisting of two X_i PUTs and two Y_i PUTs, is shown in Fig. 5, with possible bridging faults modeled as resistors.

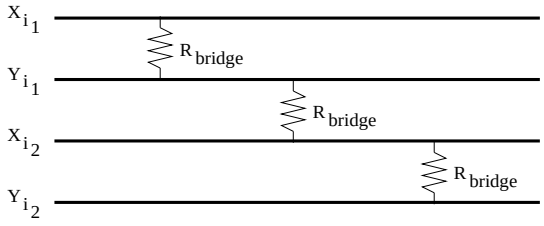


Figure 5: Interleaved Bus Signal Routing

The detection of bridging faults is based on the assumption that a bridging fault will cause a transition to occur on at least one of the bridged interconnects. A transition is deemed to have occurred when the resulting signal swing is large enough to switch the next level logic, i.e. the input gate of the destination logic block interprets a signal change. This bridging model is more general than the wired-AND and wired-OR models historically used: the only requirement is that at least one of the bridged interconnects (there may be more than two) incurs an incorrect signal transition or no signal transition occurs when one should.

For simplicity, two specific cases of the general bridging fault model are discussed: bridging faults modeled as either wired-AND or wired-OR that involve two interconnects. A bridging fault between PUTs X_{i_j} and Y_{i_k} is detected by the logic block LB_i. In test phase A (see Fig. 2a), the OR_{i_X} LUT implements the function $First_i = \mathbf{X}_i + \overline{\mathbf{Y}_i}$, with $\mathbf{X}_i$ initialized to $\mathbf{0}$ and $\mathbf{Y}_i$ initialized to $\mathbf{1}$. A wired-AND bridging fault between PUTs X_{i_j} and Y_{i_k} drives the value of the signals on X_{i_j} and Y_{i_k} to be $0 \cdot 1 = 0$ prior to the start of the actual test, which begins with the *input* signal of the ILA transitioning. The output of the OR_{i_X} LUT ($First_i$) will subsequently transition to 1 due to the bridged Y_{i_j} PUT prematurely transitioning from 1 to 0, but the output of the OR_{i_Y} LUT will remain constant because the signals of *all* PUTs have not yet correctly transitioned. The signal transition on $First_i$ causes a signal transition on $Clock_i$ that clocks flip-flops ff_{i_X} and ff_{i_Y}, making LB_i propagate an FST ($\{\mathbf{0},\mathbf{1}\}_{\mathbf{X}_{i+1},\mathbf{Y}_{i+1}} \rightarrow \{\mathbf{1},\mathbf{1}\}_{\mathbf{X}_{i+1},\mathbf{Y}_{i+1}}$) on set_{i+1}. The resulting flip-flop states of LB_i are $\{Q_{i_X}, Q_{i_Y}\} = \{1,1\}$, shown in the second row of Table 4 for configuration *A*.

Table 4: Flip-flop States for a Bridging Fault on set_i

Cfg.	Bridge Fault	Result	Resulting PUT State $\{X_{i_j}, Y_{i_k}\}$	Resulting Flip-flop State $\{Q_{i_X}, Q_{i_Y}\}$
A: init {0,1}	none	pass	0,1	0,1
	w-AND	fail	0·1=0,0	1,1
	w-OR	fail	0+1=1,1	1,1
B: init {1,0}	none	pass	1,0	1,0
	w-AND	fail	1·0=0,0	0,0
	w-OR	fail	1+0=1,1	0,0

Similarly, a wired-OR bridging fault between PUTs X_{i_j} and Y_{i_k} prematurely drives the value of the signals on X_{i_j} and Y_{i_k} to be $0 + 1 = 1$ The $First_i$ signal will subsequently transition to 1 but the $Last_i$ signal will remain constant, causing LB_i to propagate an FST on set_{i+1}. The resulting flip-flop states of LB_i are $\{Q_{i_X}, Q_{i_Y}\} = \{1,1\}$, shown as the third row of Table 4 for configuration *A*.

The values stored in ff_{i_X} and ff_{i_Y} for no bridging fault, a wired-AND bridging fault, and a wired-OR bridging fault in set_i are shown in Table 4 for test phases A and B, respectively.

6 Fault Location

Fault location is an important aspect of FPGA testing for two reasons. First, locating a fault is typically a required step in fault diagnosis, which plays an important role in modifying manufacturing processes to increase yields. Second, locating a faulty resource is crucial to the application-specific FPGA model (ASFPGA) [15]. An ASFPGA is a device in which certain resources are faulty, but remain unused for a specific customer's configuration, permitting the FPGA to function correctly with a particular design.

Locating a faulty resource in an FPGA is a generally difficult and time consuming process. Recent FPGAs (Xilinx XC4000, Virtex and Virtex-II) have an additional *design for testability* feature in which the contents of the bistable elements can be serially scanned out from the device using

special dedicated hardware. This so-called *readback* makes it possible to capture the state of an FPGA at a particular instant in time, making the fault location of a configured FPGA possible through numerous iterative steps [16]. Such methods, using a binary search, require $O\lceil log_2 M \rceil$ FPGA reconfigurations (re-initializations of the FPGA) and readbacks to locate a fault in an M-length ILA. For example, to locate a stuck-at 0 or stuck open fault (or perhaps a bridging or delay fault) in a clocked ILA of M bistable elements, a single 1 is serially shifted down the ILA (a single 0 is shifted to locate a stuck-at 1 fault). In this common fault location method, one readback must be obtained at each of the $\lceil log_2 M \rceil$ decision steps to determine where the single shifted 1 (or 0) fails to shift down the ILA. If the 1 is not present in the ILA after shifting for some number of clock cycles (as seen by the state of the appropriate bistable element via readback), the FPGA is reconfigured (all the bistable elements must be reinitialized to 0) and the FPGA must be retested with a reduced number of clock cycles being applied to the ILA. If the 1 is still present, the FPGA must be retested with an increased number of clock cycles being applied to the ILA.

The presented method stores the test result of each set_i in the flip-flops of LB_i ($\{Q_{i_X}, Q_{i_Y}\}$). The first failing set, say set_i, causes the final value of the FST to be stored in the logic blocks LB_i through LB_{M-1} ($\{1,1\}$ for test phase A and $\{0,0\}$ for test phase B). Therefore, only one readback is required to determine the faulty set, corresponding to the first logic block storing the failing-result value. For example, given an M-PUT ILA, a readback of the ILA's flip-flops is written as $< \{Q_{0_X}, Q_{0_Y}\}\{Q_{1_X}, Q_{1_Y}\} \ldots \{Q_{M_X}, Q_{M_Y}\} >$. For test phase A, the readback contains $< \{0,1\}\{0,1\} \ldots \{0,1\} >$ before testing (and in the absence of any bridging faults between PUTs), $< \{1,0\}\{1,0\} \ldots \{1,0\} >$ after testing if the FPGA passes, and $< \{1,0\}\{1,0\} \ldots \{1,0\}\{1,1\} \ldots \{1,1\} >$ after testing if the FPGA fails. The readback corresponding to the failing FPGA shows that LB_i is the first logic block storing the failing value ($\{1,1\}$), thus quickly determining that one of the $2N$ PUTs of set_i has a delay fault.

7 Conclusion

The presented interconnect delay fault testing method achieves an adjustably high sensitivity to delay faults, and is capable of multiple-fault detection. The method is applicable to all SRAM-based FPGAs, independent of both device family and vendor, and scales with FPGA array size. Using an RC model, it is shown for several FPGA device families that resistive open defects of several kilo-ohms are detectable. Detection of single or multiple bridging faults is also realized. Additionally, fault location is quickly and easily accomplished by scanning out the contents of the bistable elements after testing.

Acknowledgment

This work is supported by Xilinx, Inc. under contract no. 2DSA907. The author thanks Prof. Edward McCluskey for his valuable guidance and Dr. Shahin Toutounchi for providing helpful information about Xilinx FPGAs.

References

[1] M. Renovell and Y. Zorian, "Different experiments in test generation for XILINX FPGAs," *Proc. Int. Test Conf.*, pp. 854–862, 2000.

[2] C. Stroud, S. Konala, C. Ping, and M. Abramovici, "Built-in self-test of logic blocks in FPGAs (finally, a free lunch: BIST without overhead)," *Proc. VLSI Test Symp.*, pp. 387–392, 1996.

[3] W. Huang and F. Lombardi, "An approach to testing programmable/configurable field programmable gate arrays," *Proc. VLSI Test Symp.*, pp. 450–455, 1996.

[4] X. Sun, S. Xu, J. Xu, and P. Trouborst, "Design and implementation of a parity-based BIST scheme for FPGA global interconnects," *Canadian Conf. Electrical and Computer Engineering*, pp. 1251–1257, 2001.

[5] A. Doumar and H. Ito, "Testing the logic cells and interconnect resources for FPGAs," *Proc. Eighth Asian Test Symp.*, pp. 369–374, 1999.

[6] M. Renovell, J. Portal, J. Figueras, and Y. Zorian, "Testing the interconnect of RAM-based FPGAs," *IEEE Design and Test of Computers*, pp. 45–50, 1998.

[7] S. Wang and C. Huang, "Testing and diagnosis of interconnect structures in FPGAs," *Proc. Seventh Asian Test Symp.*, pp. 283–287, 1998.

[8] M. Renovell, J. Figueras, and Y. Zorian, "Test of RAM-based FPGA: Methodology and application to interconnect," *Proc. VLSI Test Symp.*, pp. 230–237, 1997.

[9] H. Michinishi, T. Yokohira, and T. Okamoto, "A test methodology for interconnect structures of LUT-based FPGAs," *Proc. IEEE Asian Test Symp.*, pp. 68–74, 1996.

[10] M. Tahoori, "Improving detectability of resistive open defects in FPGA," *MAPLD Int. Conf.*, 2002.

[11] M. Abramovici and C. Stroud, "BIST-based delay-fault testing in FPGAs," *Proc. Eighth IEEE Int. On-Line Testing Workshop*, pp. 131–134, 2002.

[12] I. Harris, P. Menon, and R. Tessier, "BIST-based delay path testing in FPGA architectures," *Proc. Int. Test Conf.*, pp. 932–938, 2001.

[13] A. Krasniewski, "Testing FPGA delay faults in the system environment is very different from ordinary delay fault testing," *Proc. Seventh Int. On-line Testing Workshop*, pp. 37–40, 2001.

[14] W. Needham and E. Prunty, "High volume microprocessor test escapes, an analysis of defects our tests are missing," *Proc. Int. Test Conf.*, pp. 25–34, 1998.

[15] Xilinx, Inc., "Xilinx easypath solutions." http://www.xilinx.com/xlnx/xil_prodcat_product.jsp?title=v2_easypath, 2003.

[16] K. Tomko and A. Tiwari, "Hardware/software co-debugging for reconfigurable computing," *High Level Validation and Test Workshop*, pp. 59–63, 2000.