Center for
Reliable
Computing

‘Path Phase Difference’ Delay Fault Testing of Routing Resources in FPGAs with Diagnosis

Erik Chmelar
Center for Reliable Computing
Stanford University
January 27, 2003

Outline

- **Introduction**
- Objective
- Implementation
- Analysis
- Fault Coverage
- Diagnosis
- Conclusion
- Future Work

Introduction

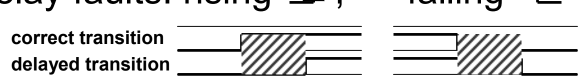
Motivation

- ~80% of die area is routing resources
  - wire segments, PSMs, PIPs, etc.
- VLSI scaling causing more delay defects [Abr. & Str. 02]
- Delay defects can cause reliability issues
  - Resistive opens can become ‘worse’
    - metal migration

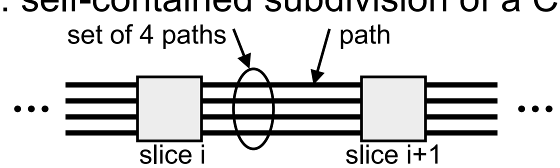
Introduction

Definitions

- Delay faults: rising, falling



- Path: group of elements adjoining two slices
  - wire segments, vias, PSMs, PIPs
- Set: group of paths simultaneously tested
- Slice: self-contained subdivision of a CLB



Introduction

Basic Idea

- Path delay α $(R^*_{path} + R^*_{def}) \times (C^*_{path} + C^*_{def})$
- Test for delay defects
  – Excess resistance (resistive open)
  – Excess capacitance (capacitive coupling)

Introduction

Regularity of Routing Resources

- Routing resources are bus oriented
  – Usually 4-8 wires per bus
  – Allows creation of signal paths with nearly identical propagation delays
- Exploit regularity to test for delay faults

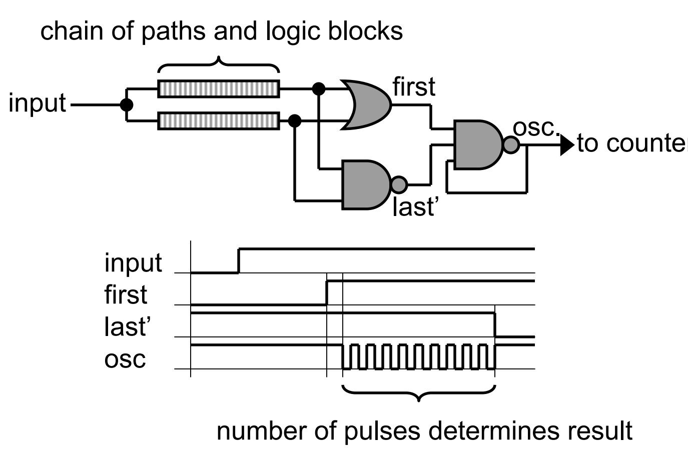
Introduction

Previous Work [Abr. & Str. 02]

- Configure chains of paths and logic blocks
  - Same sequence of LUTs, latches, wire segments, etc. (all transparent, no clock)
- Apply input transition
- Measure phase difference at output by an oscillator loop and counter
  - Determine some threshold number of oscillations

Introduction

Previous Work [Abr. & Str. 02]



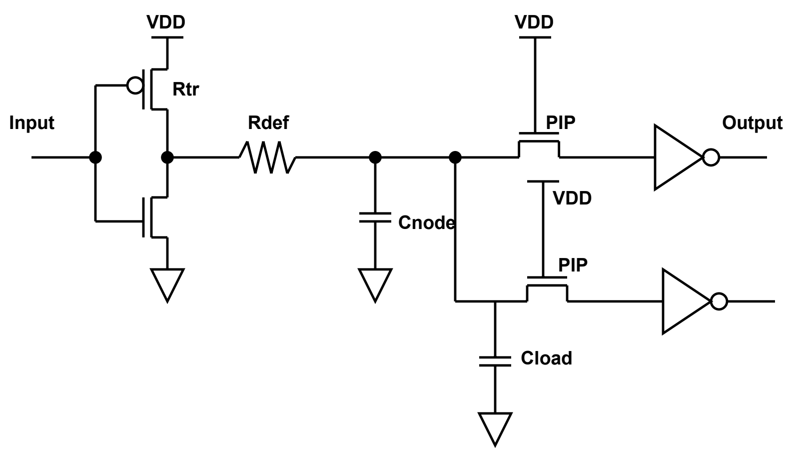
Introduction

Previous Work [Tahoori 02]

- Configure chains of paths and logic blocks
  - Add additional load to path
    - close one or more additional PIPs
- Shift 1 (0) synchronously through chain of 0’s (1’s)
  - Determine where 1 (0) is ‘lost’
  - Delay α $[R_{tr}(V_{DD}) + R_{defect}] \cdot [C_{node} + C_{load}]$
    - additional load increases delay of path

Introduction

Previous Work [Tahoori 02]



Outline

- Introduction
- **Objective**
- Implementation
- Analysis
- Fault Coverage
- Diagnosis
- Conclusion
- Future Work

Objective

Delay Fault Test Criteria

- Detect delay defects in routing resources
  - Few configurations
    - fast test time
  - Diagnosable
    - ability to quickly localize failure location
  - Ability to detect other faults
    - stuck-at 0/1
    - stuck-open
    - bridging (wire AND/OR)

Outline

- Introduction
- Objective
- **Implementation**
- Analysis
- Fault Coverage
- Diagnosis
- Conclusion
- Future Work

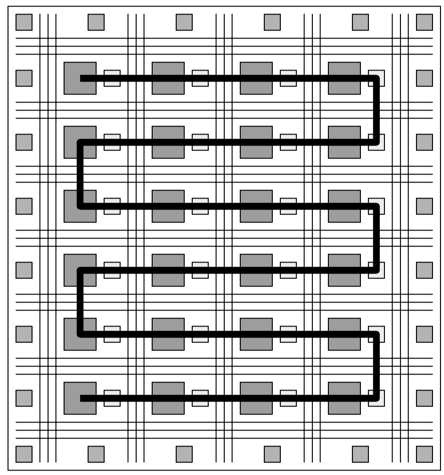
Implementation

Overview

- Configure FPGA into set/slice chains
  - Slice output tests (controls) next set
  - Slice input detects if delay defect present in previous set of paths
    - measures paths' signal phase difference
      - tests next set if previous passed
      - propagates failure if previous failed
- Apply single input transition (no clock)
- Observe single output transition

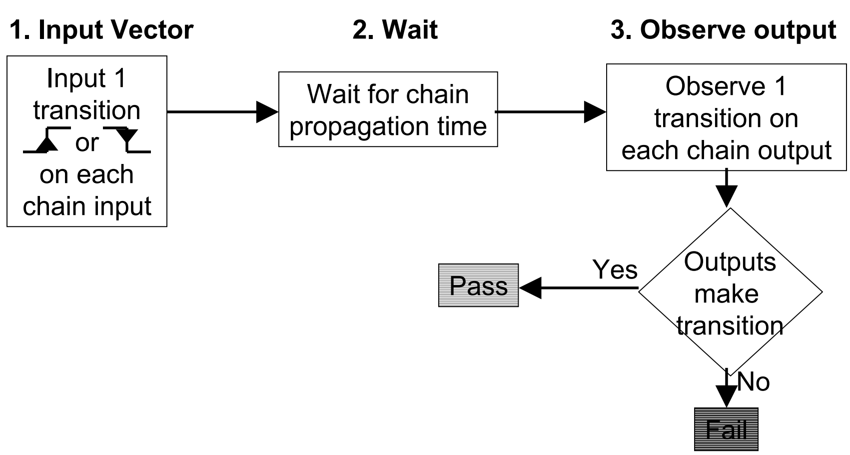
Implementation

Chain Signal Flow



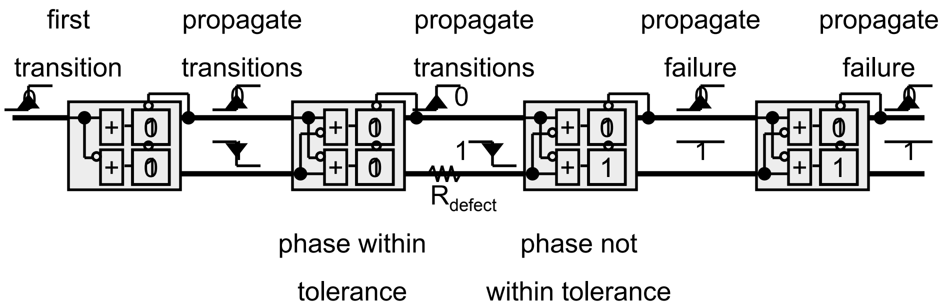
Implementation

Test Flow Chart



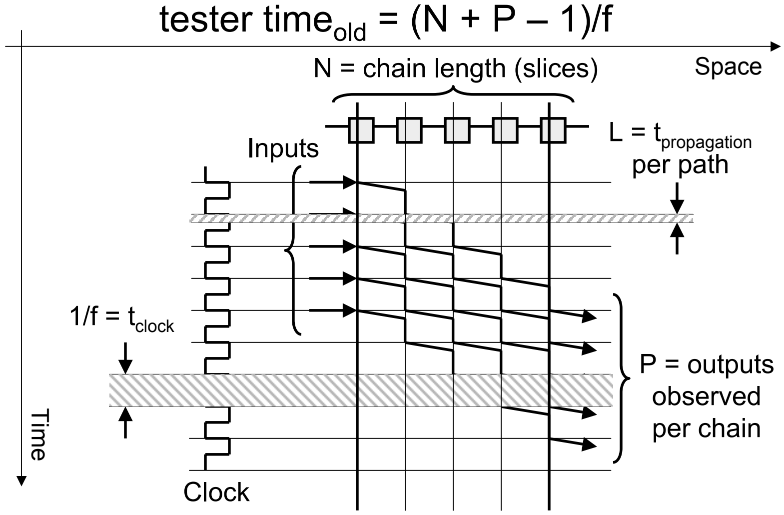
Implementation

Chain Propagation Example (Cfg. A)



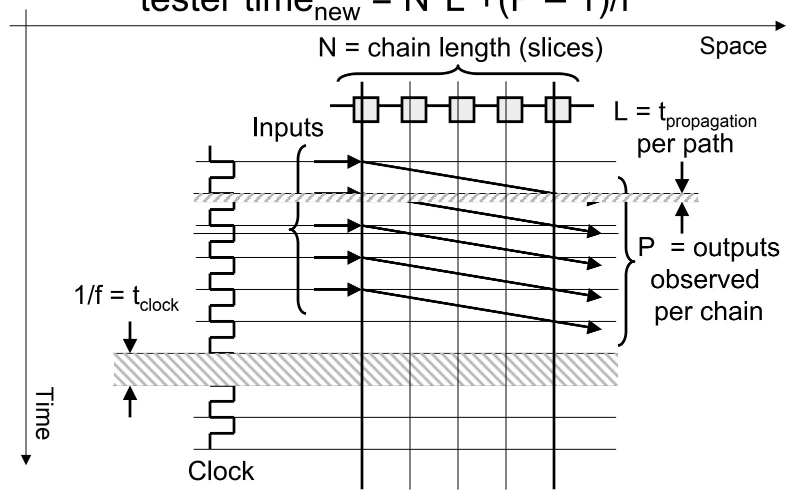
Implementation

tester $time_{old} = (N + P - 1)/f$



Implementation

tester $time_{new}$ = N*L +(P – 1)/f



Implementation

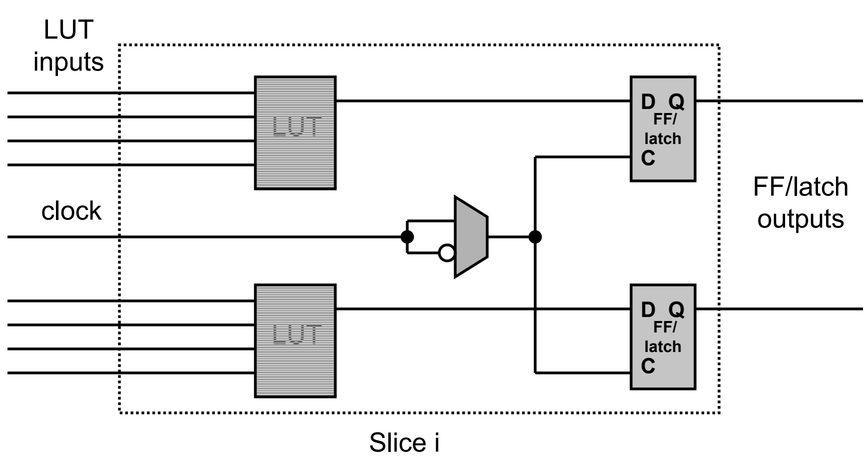
Pattern Application Speedup Example

- speedup = test $time_{old}$/test $time_{new}$ ≈ 1/(f*L)
  - f = 60 MHz (ATE clock)
  - P = 1 (observe 1 output cycle)
  - N = 10000 (chain length (slices))
  - L = 2 ns (path length (delay))
  - test $time_{old}$ = (N + P – 1)/f = 167 μs
  - test $time_{new}$ = N*L + (P – 1)/f = 20 μs
  - speedup ≈ 8

Implementation
Configurations

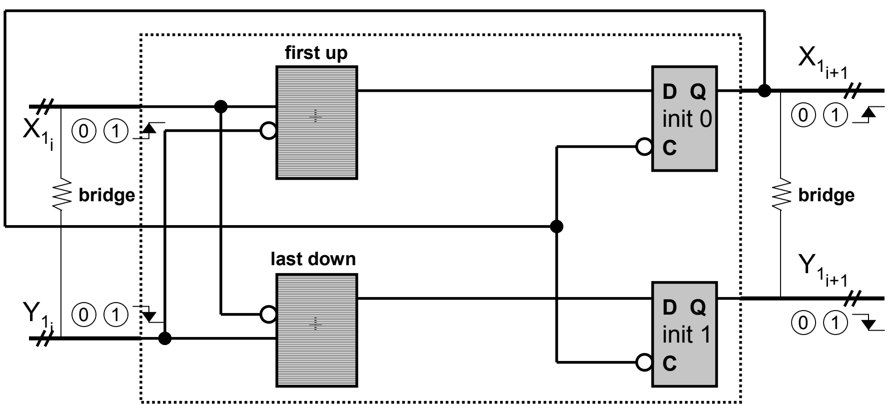
- Two configurations to test each set
  - One for _/‾ delay, one for ‾_ delay
  - 2 ≤ paths in set ≤ maximum inputs to LUT
    - at least 2 paths, each races the other
    - typically 4 inputs per LUT, 2 LUTs per slice
      - half the set is input to each LUT

Implementation
Simple View of a Slice



Implementation

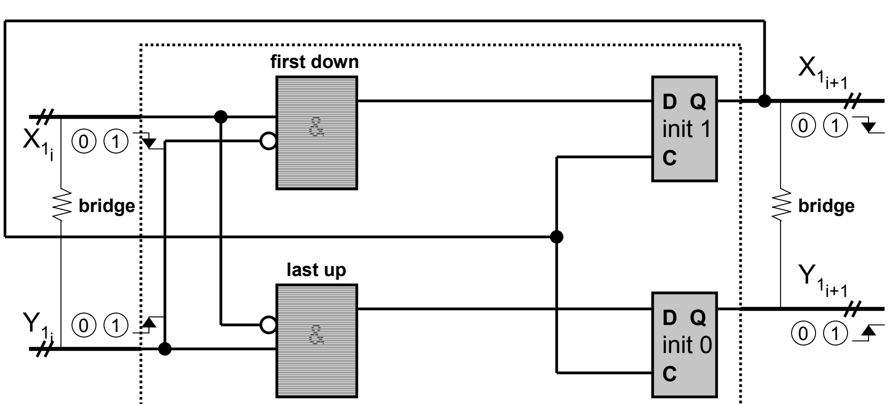
Configuration A: X, Y



Slice i

Implementation

Configuration B: X, Y



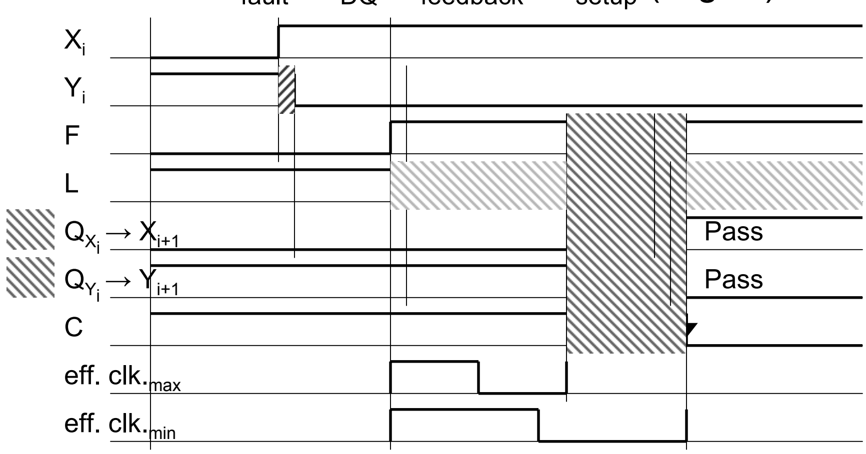
Slice i

Outline

- Introduction
- Objective
- Implementation
- **Analysis**
- Fault Coverage
- Diagnosis
- Conclusion
- Future Work

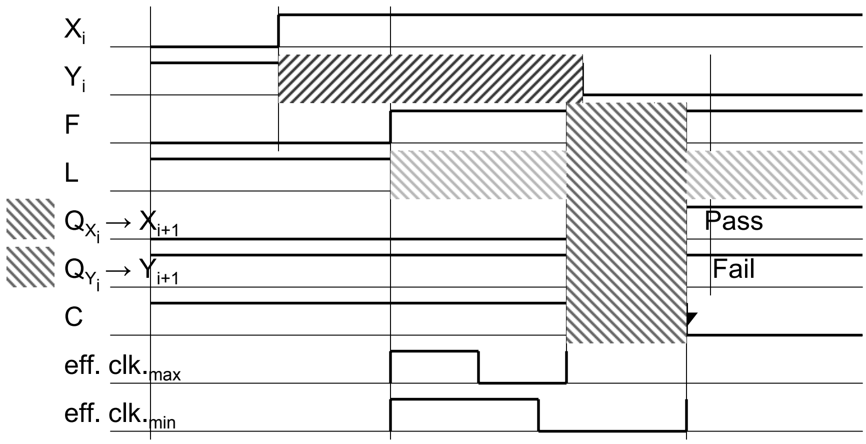
Analysis

$$0 \le t_{fault} \le t_{DQ} + t_{feedback} - t_{setup} \text{ (Cfg. A)}$$



Analysis

$t_{fault} \geq t_{DQ} + t_{feedback} + t_{hold}$ (Cfg. A)



Analysis

Xilinx Virtex-II Timing Values[1]

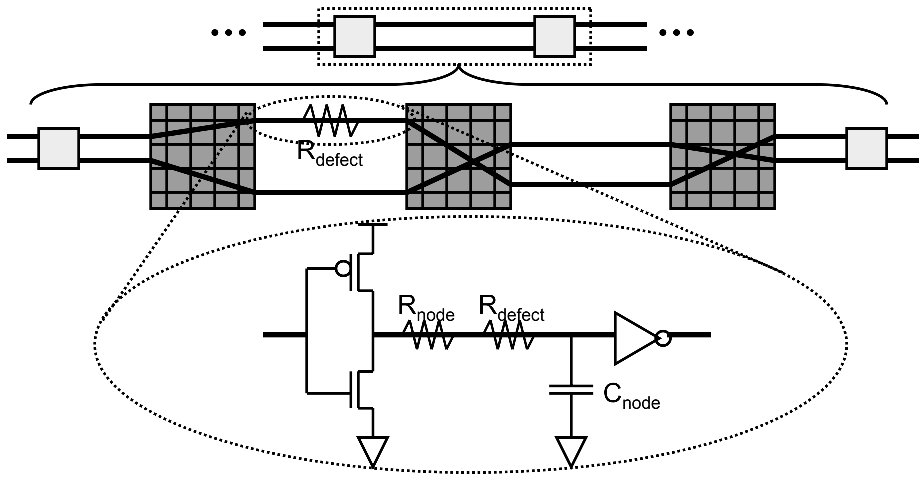
$$t_{fault_{max-pass}} = t_{DQ} + t_{feedback} - t_{setup}; \quad t_{fault_{min-fail}} = t_{DQ} + t_{feedback} + t_{hold}$$

	Speed Grade			
Symbol	-6 (fastest)	-5	-4 (slowest)	Units
[2]$t_{DQ\text{-}latch}$	.54	.59	.68	ns, max
t_{setup}	.30	.33	.37	ns, max
$t_{feedback}$	.1	.1	.1	ns, approx
t_{hold}	-.07	-.08	-.09	ns, min
$t_{fault\text{-}max\text{-}pass}$	.34	.36	.41	ns, max
$t_{fault\text{-}min\text{-}fail}$	.57	.61	.69	ns, min
eff. $clk._{max}$	2.94	2.78	2.44	GHz, max

[Xilinx 01] [1]Other device families/vendors are similar; [2]$t_{DQ} = t_{CQ}$

Analysis

Intermediate Nodes



Analysis

$$V(t) = V_{DD}(1 - e^{t/R^*_{path}C^*_{path}}); \quad \text{let } R^*,\ C^* = R,\ C$$

$$t_{50\%} = -(R_{path} + R_{defect})C_{path}\ln(.5)$$

$$t_{fault} = t_{50\%_{faulty}} - t_{50\%_{fault-free}} = -R_{defect}C_{node}\ln(.5)$$

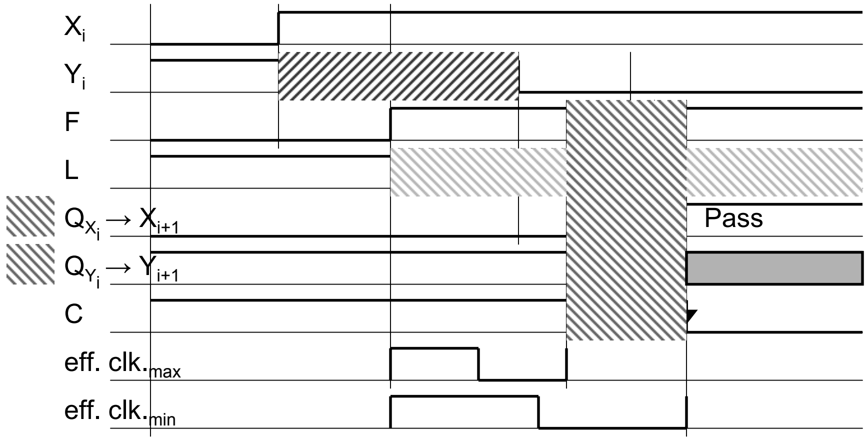
$$R_{defect_{max-pass}} = \frac{-t_{fault_{max-pass}}}{C_{node}\ln(.5)}; \quad R_{defect_{min-fail}} = \frac{-t_{fault_{min-fail}}}{C_{node}\ln(.5)}$$

	Speed Grade			
Symbol	-6 (fastest)	-5	-4 (slowest)	Units
[1]$R_{defect-pass-max}$	1.0	1.1	1.2	kΩ, max
[1]$R_{defect-fail-min}$	1.6	1.8	2.0	kΩ, min
[2]$R_{defect-fail-approx}$	48			kΩ, approx

[1]C_{node} approximated as .5 pF; [2]Assume f = 60 MHz tester clock

Analysis

$(t_{DQ} + t_{feedback} - t_{setup}) \le t_{fault} \le (t_{DQ} + t_{feedback} + t_{hold})$ (Cfg. A)



Analysis

Chain Propagation

Cfg.	Delay Fault Presence	Test Result	Test Detection	Diagnosis Readback File Q_{X_k} Q_{Y_k} k < i	k = i	k > i
A: init 01	none	pass	n/a	10	10	10
	marginal	pass	no	10	10	10
		fail	yes	10	11	11
	gross	fail	yes	10	11	11
B: init 10	none	pass	n/a	01	01	01
	marginal	pass	no	01	01	01
		fail	yes	01	00	00
	gross	fail	yes	01	00	00

Outline

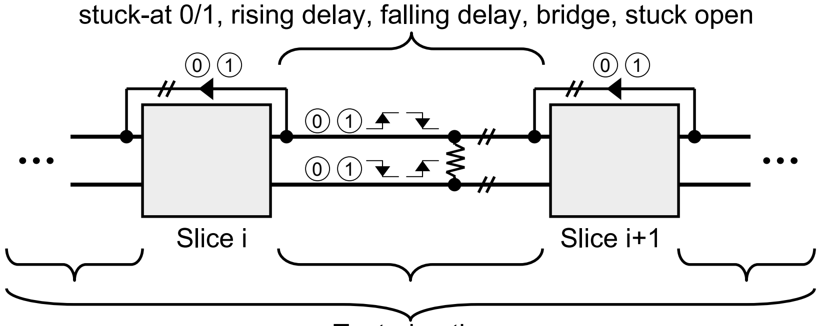
- Introduction
- Objective
- Implementation
- Analysis
- **Fault Coverage**
- Diagnosis
- Conclusion
- Future Work

Fault Coverage

Interconnect Fault Coverage

Tested faults:

stuck-at 0/1, rising delay, falling delay, bridge, stuck open



Tested paths:

wires, vias, PSMs, PIPs

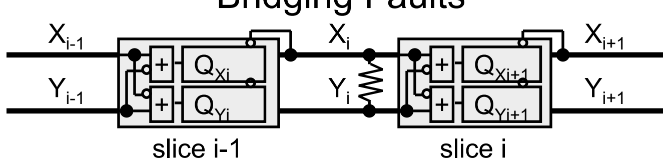
Fault Coverage

Interconnect Fault Coverage

- Rising/falling delay fault
- Stuck-at 0/1 (infinite delay)
  - Never any transition on path
- Stuck open
  - Probably never any transition on path
    - if coupling to adjacent path, then behaves similar to bridge
- Bridge
  - Opposite polarity on paths detect bridge

Fault Coverage

Bridging Faults

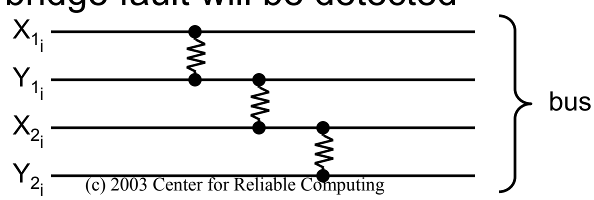


Cfg.	Bridging Fault	$X_i(k)$ $Y_i(k)$	$X_i(k+1)$ $Y_i(k+1)$	$X_{i+1}(k+1)$ $Y_{i+1}(k+1)$	$X_{i+1}(k+2)$ $Y_{i+1}(k+2)$	Test Result
A: init 01	none	01	10	01	10	pass
	wire AND	0·1 = 00	1·0 = 00	11	11	fail
	wire OR	0+1 = 11	1+0 = 11	11	11	fail
B: init 10	none	10	01	10	01	pass
	wire AND	1·0 = 00	0·1 = 00	00	00	fail
	wire OR	1+0 = 11	0+1 = 11	00	00	fail

Fault Coverage

Bridging Fault Bus Line Staggering

- If set consists of more than two paths
  - Stagger X and Y lines
    - X, Y always driven with opposite polarity
    - bridge fault (wire AND/OR) causes X, Y lines to always have same value
      - bridge fault will be detected



Outline

- Introduction
- Objective
- Implementation
- Analysis
- Fault Coverage
- **Diagnosis**
- Conclusion
- Future Work

Diagnosis

Standard Shift Chain Testing

- Shift single 1 (0) through chain of 0’s (1’s)
  - Determine where 1 (0) is ‘lost’
    - Iterate:
      - configure
      - test
      - readback
      - determine if 1 (0) is ‘lost’
      - modify test length

Diagnosis

Single Oscillator Loop [Abr. & Str. 02]

- No reasonable diagnosis method
  - All paths grouped together in long chains
    - no intermediate path test results stored
  - Iterate:
    - configure
    - test
    - count oscillations to determine if faulty
    - re-route chains (eliminate LUTs, wire segments, etc.)

Diagnosis

Path Phase Difference

- No search iterations necessary
  - Incremental test result saved in each slice
  - Configure
  - Test
  - Observe output transition
  - XOR readback with expected value
    - first non-zero entry determines failing set
      - identifies set between 2 slices

Diagnosis

Path Phase Difference Example (Cfg. A)

readback file:		$Q_{X_0}Q_{Y_0}$	$Q_{X_1}Q_{Y_1}$	...	$Q_{X_{i-1}}Q_{Y_{i-1}}$	$Q_{X_i}Q_{Y_i}$	$Q_{X_{i+1}}Q_{Y_{i+1}}$	...	$Q_{X_N}Q_{Y_N}$
before test:		01	01	...	01	01	01	...	01
test passed:		10	10	...	10	10	10	...	10
test failed:	$\oplus$	10	10	...	10	11	11	...	11
		00	00	...	00	01	01	...	01
		set_0	set_1	...	set_{i-1}	set_i	set_{i+1}	...	set_N

determines failing set of paths as set i

Outline

- Introduction
- Objective
- Implementation
- Analysis
- Fault Coverage
- Diagnosis
- **Conclusion**
- Future Work

Conclusion

- Able to detect small delay defects (~1-2 kΩ)
- Also detect stuck-at 0/1, stuck open, bridge
- Able to quickly localize faulty set of paths
- Test pattern application time reduced

Outline

- Introduction
- Objective
- Implementation
- Analysis
- Fault Coverage
- Diagnosis
- Conclusion
- **Future Work**

Future Work

- Try out method on actual devices
  - Are the timing thresholds appropriate?
    - $t_{fault_{pass\text{-}max}} = t_{DQ} + t_{feedback} - t_{setup}$
    - $t_{fault_{fail\text{-}min}} = t_{DQ} + t_{feedback} + t_{hold}$
    - don’t want threshold too low
      - false failures
  - Will setup time violation cause problems?
    - metastable oscillations

References (1)

[Abr. & Str. 02] Abramovici, M and Stroud, C. “BIST-Based Delay-Fault Testing in FPGAs,” Proc. of the Eighth International On-Line Testing Workshop, 131-134, 2000.

[Tahoori 02] Tahoori, M. B. “Testing for Resistive Opens in FPGA,” Center for Reliable Computing Technical Report, 2002.

[Xilinx 01] Xilinx, Inc., San Jose, Ca. “Virtex-II Platform FPGA Handbook,” December 2001.

