The Benefits of Feedback TMR for SEU Tolerance of SRAM FPGA Designs

Mike Wirthlin

BYU

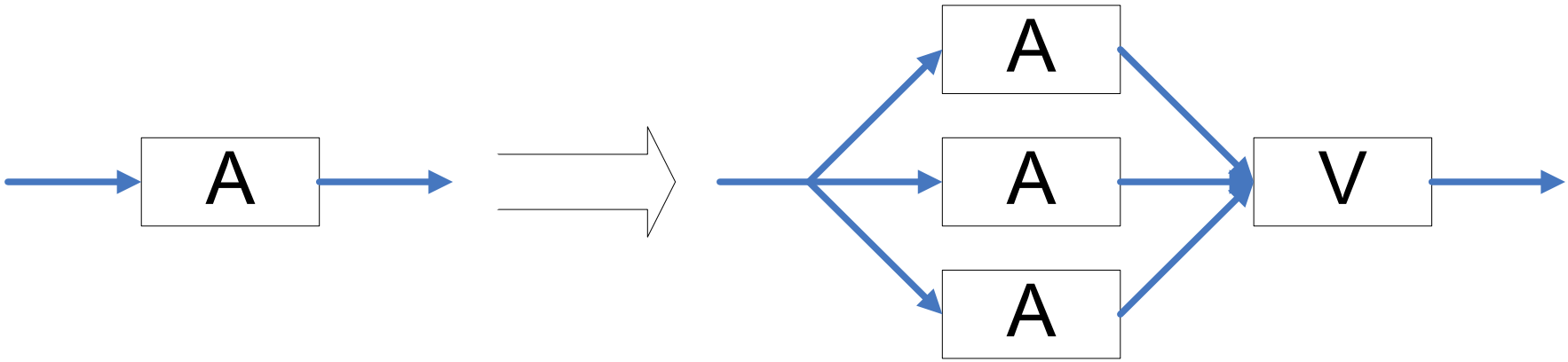
Provo, Utah USA

SEFUW: SpacE FPGA Users Workshop, March 2016

* This work was sponsored by the Department of Energy, Los Alamos National Laboratory under contract #95952-001-04 3C, the National Science Foundation I/UCRC Center for High Performance Reconfigurable Computing (CHREC) under contracts #0801876 and #1265957, and Cisco Systems.

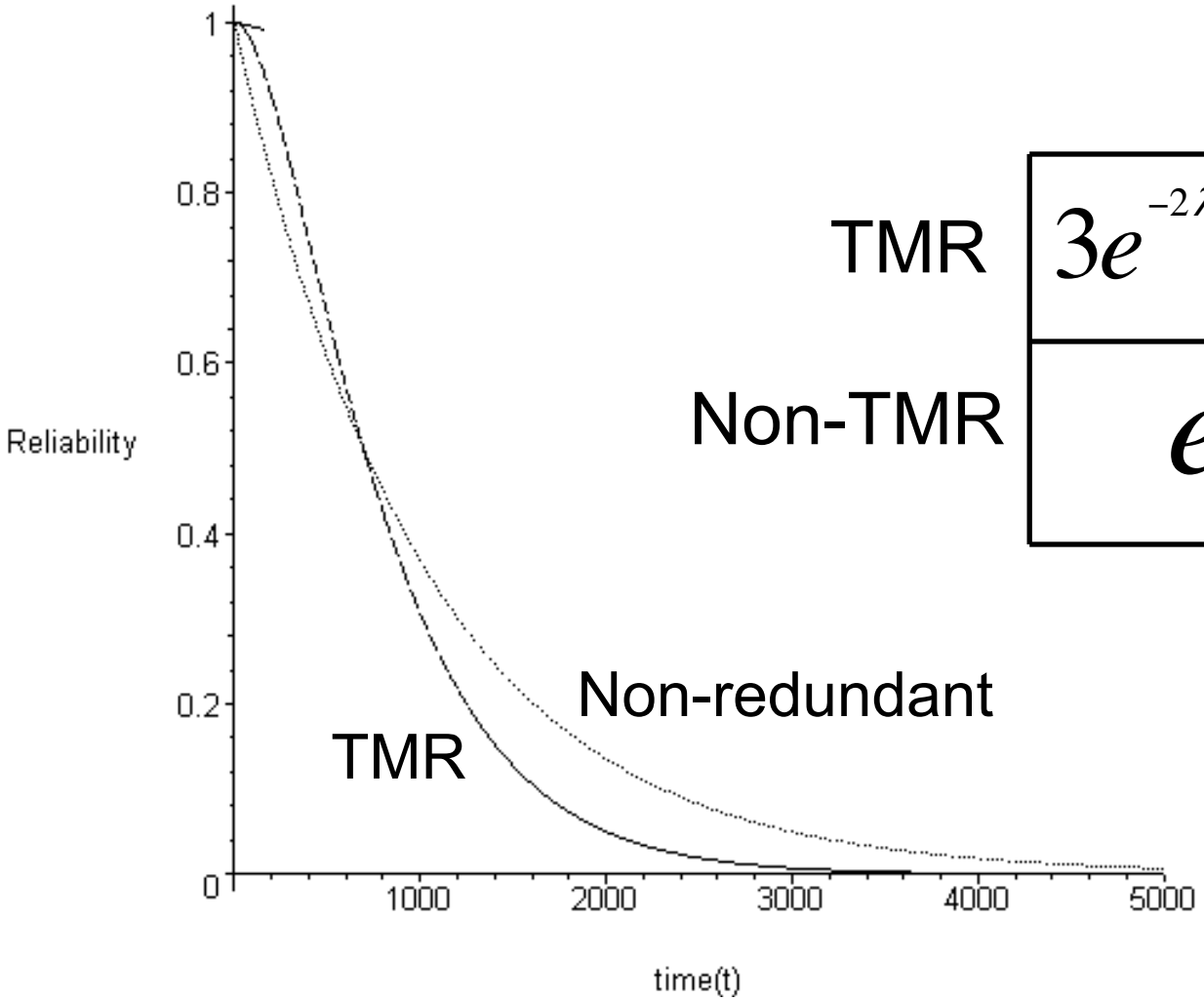
Triple Modular Redundancy (TMR)

- A form of N Modular Redundancy
  - *Triplicate hardware resources*
  - *Majority Vote on hardware outputs*



- Tolerates any *single* fault
  - *Tolerates many multiple fault combinations*

TMR Reliability

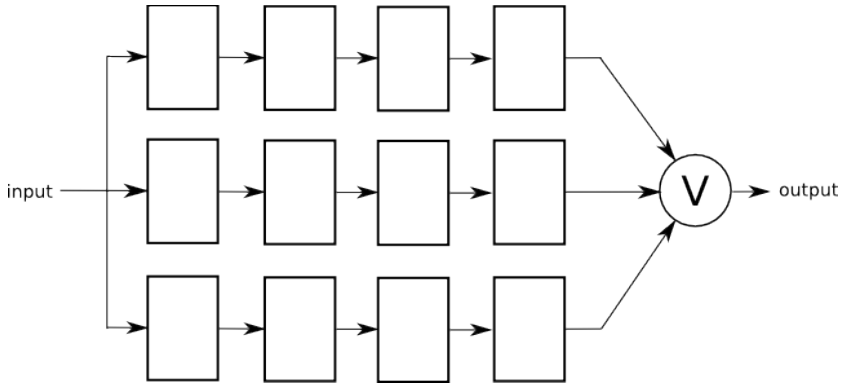


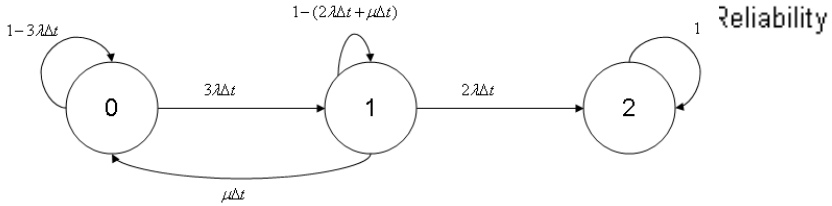
	R(t)	MTTF
TMR	$3e^{-2\lambda t} - 2e^{-3\lambda t}$	$5/6\lambda$
Non-TMR	$e^{-\lambda t}$	$1/\lambda$

λ = failure rate

TMR has *lower* MTTF than non-redundant systems

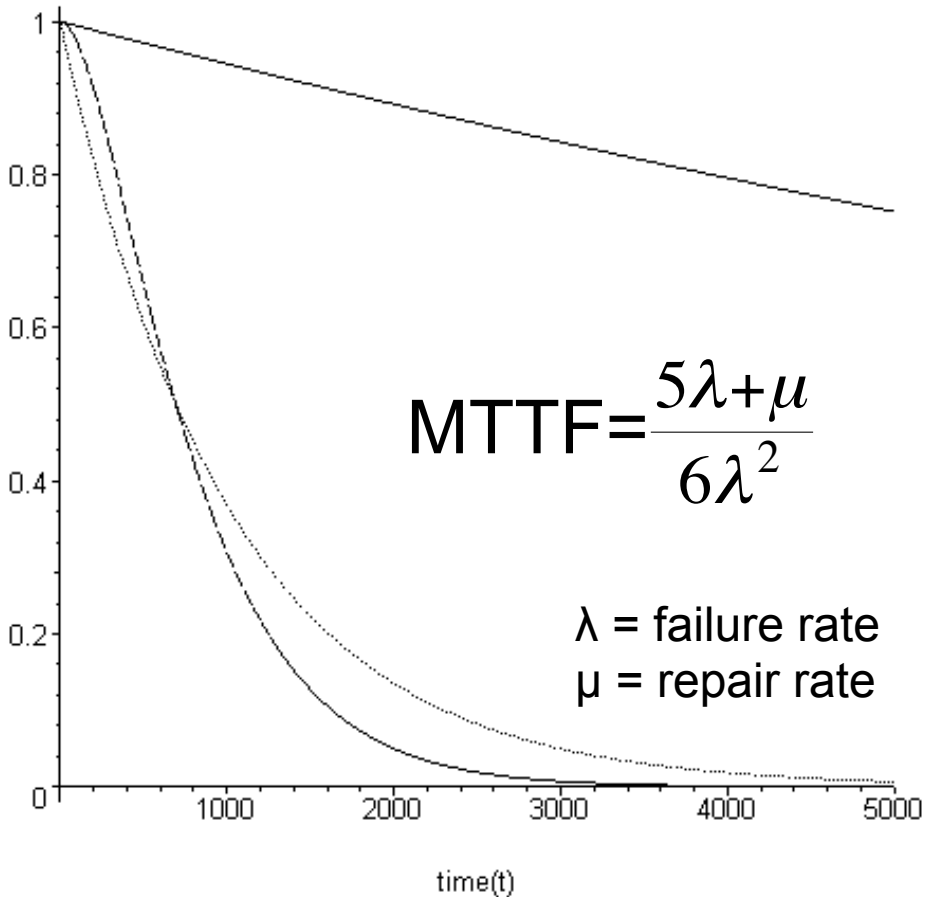
TMR + Repair = Very Reliable!





$$R(t) = 1 - p_2(t) \quad (44)$$

$$= \frac{(\mu+5\lambda)sinh(\frac{1}{2}t\sqrt{\mu^2+10\lambda\mu+\lambda^2})e^{-\frac{1}{2}(\mu+5\lambda)t}}{\sqrt{\mu^2+10\lambda\mu+\lambda^2}} + \cosh(\frac{1}{2}t\sqrt{\mu^2+10\lambda\mu+\lambda^2})e^{-\frac{1}{2}(\mu+5\lambda)t} \quad (45)$$



$$MTTF = \frac{5\lambda+\mu}{6\lambda^2}$$

λ = failure rate
μ = repair rate

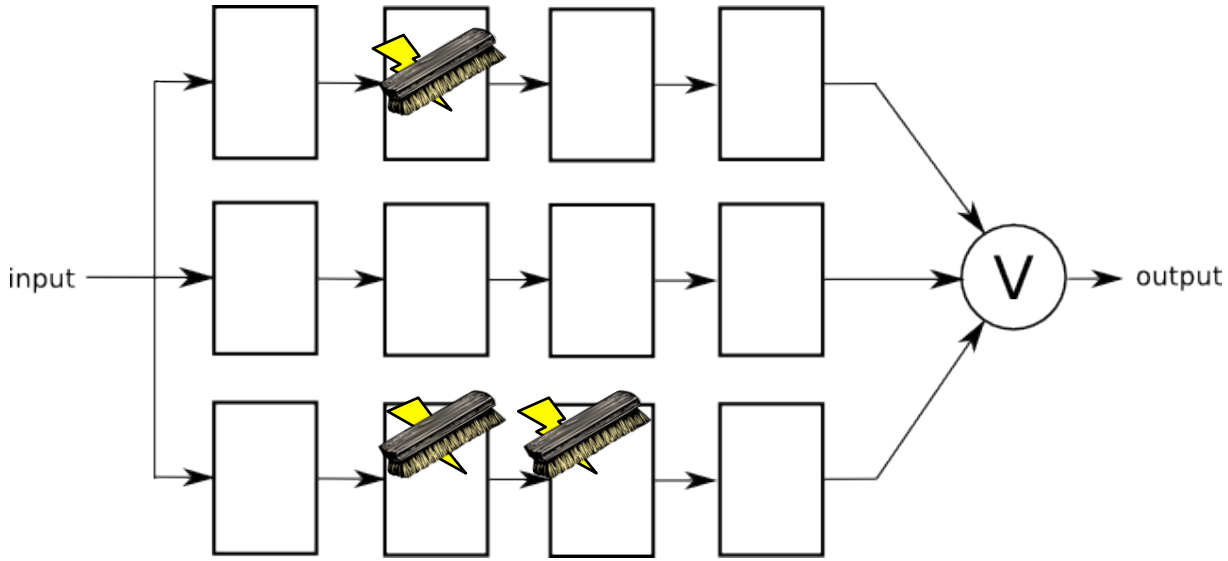
Using Classical Reliability Models and Single Event Upset (SEU) Data to Determine Optimum Implementation Schemes for Triple Modular Redundancy (TMR), **M. D. Berg**, H. S. Kim, C. M. Seidleck, A. M. Phan, K. A. LaBel, J. Pellish, M. J. Campolla

SRAM FPGA Reliability Analysis for Harsh Radiation Environments, **P.S. Ostler**, M. P. Caffrey ; D. S. Gibelyou ; P. S. Graham ; K. S. Morgan ; B. H. Pratt ; H. M. Quinn ; M. J. Wirthlin, IEEE TNS, vol 56, no 6, pp. 3519-3526, Dec. 2009.

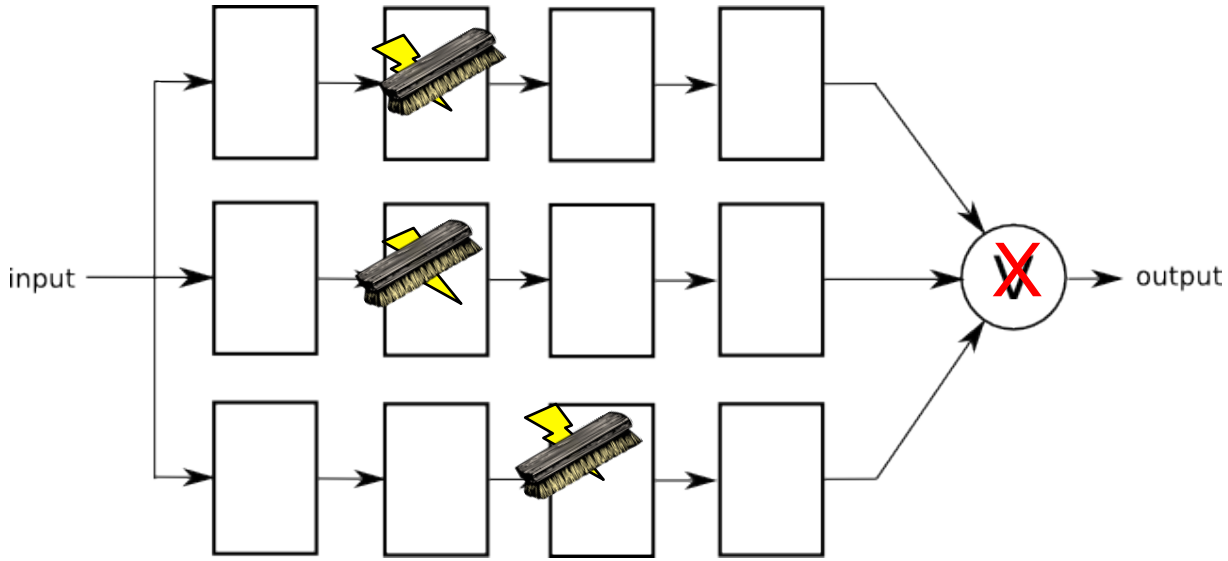
FPGA "Repair"

- "Repair" configuration memory
  - *Replace single-event upset in configuration memory with "correct" value*
  - *Configuration Scrubbing*
    - Continuously monitor and write configuration data
    - Partial reconfiguration
    - Many strategies and techniques for scrubbing
- Resynchronization
  - *Restore the operating state of the failed circuit to the state of the correct circuits*
  - *Can be challenging in real time*

TMR & Scrubbing Example



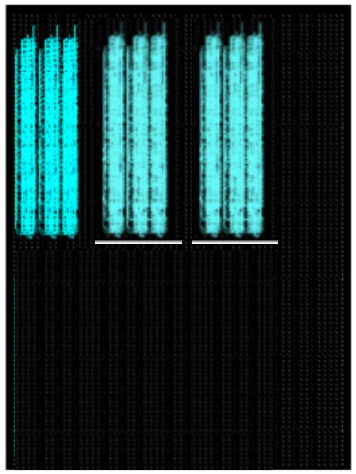
TMR & Scrubbing Example



TMR Granularity

System Level

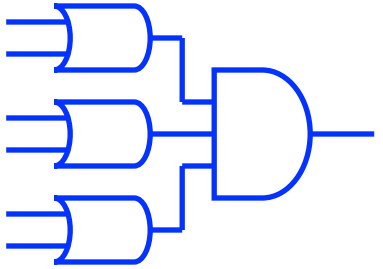
Device Level



Module Level

```
process(clk_int_a)
begin
  if clk_int_a'event and clk_int_a='1' then
     locked_d_a <= locked_a_int;
     if (all_locked_a = '0') then
       all_locked_a <= (locked_d_a and
            locked_d_b and locked_d_c);
     else
        all_locked_a <= tmr_voter(
          locked_d_a, locked_d_b,
          locked_d_c);
     end if;
  end if;
end process
```

RTL Level



Logic Level

TMR Automation

- Limitations of manual application of TMR
  - *Tedious design process*
  - *Error prone (improper TMR application, design errors)*
  - *Must redesign circuit each time TMR approach changes*
- TMR is relatively easy to automate
  - *Analyze design*
  - *Replicate resources*
  - *Insert voters*
  - *Verify resulting circuit*
- Different Strategies for Automated TMR
  - *Netlist level*
  - *HDL Level*
  - *Selective/Partial*
- Several tools available for Automatic TMR

Automated TMR Tools

TMRTool

XILINX®

Precision® Hi-Rel

Mentor Graphics®

SYNPLIFY PREMIER
The Ultimate FPGA Implementation Platform

SYNOPSYS®

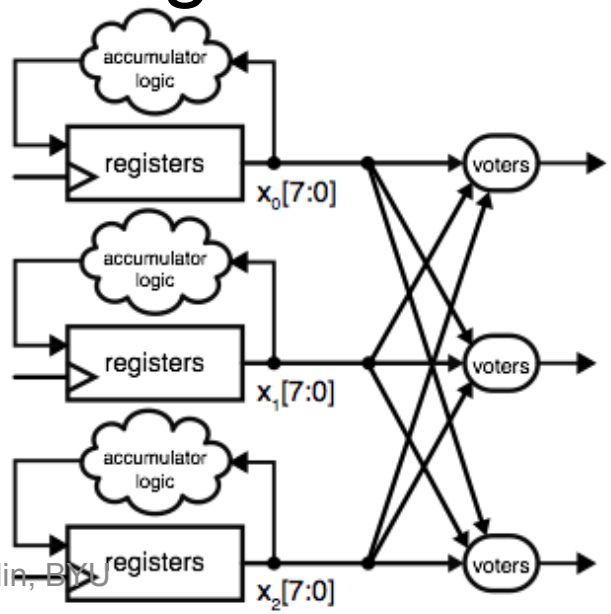
BL-TMR

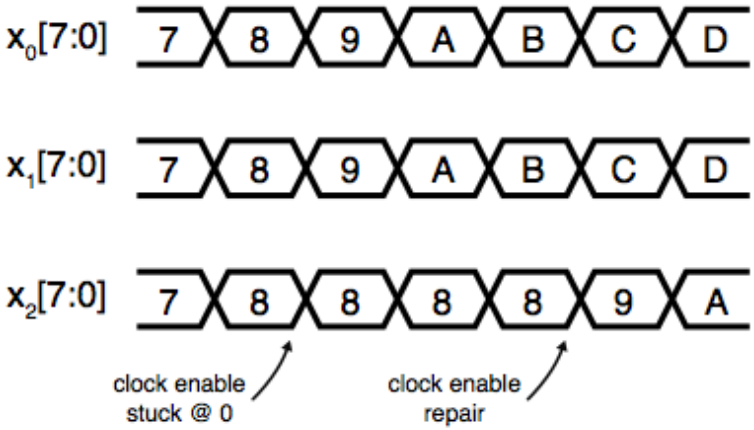
BYU™

(and other several other academic projects)

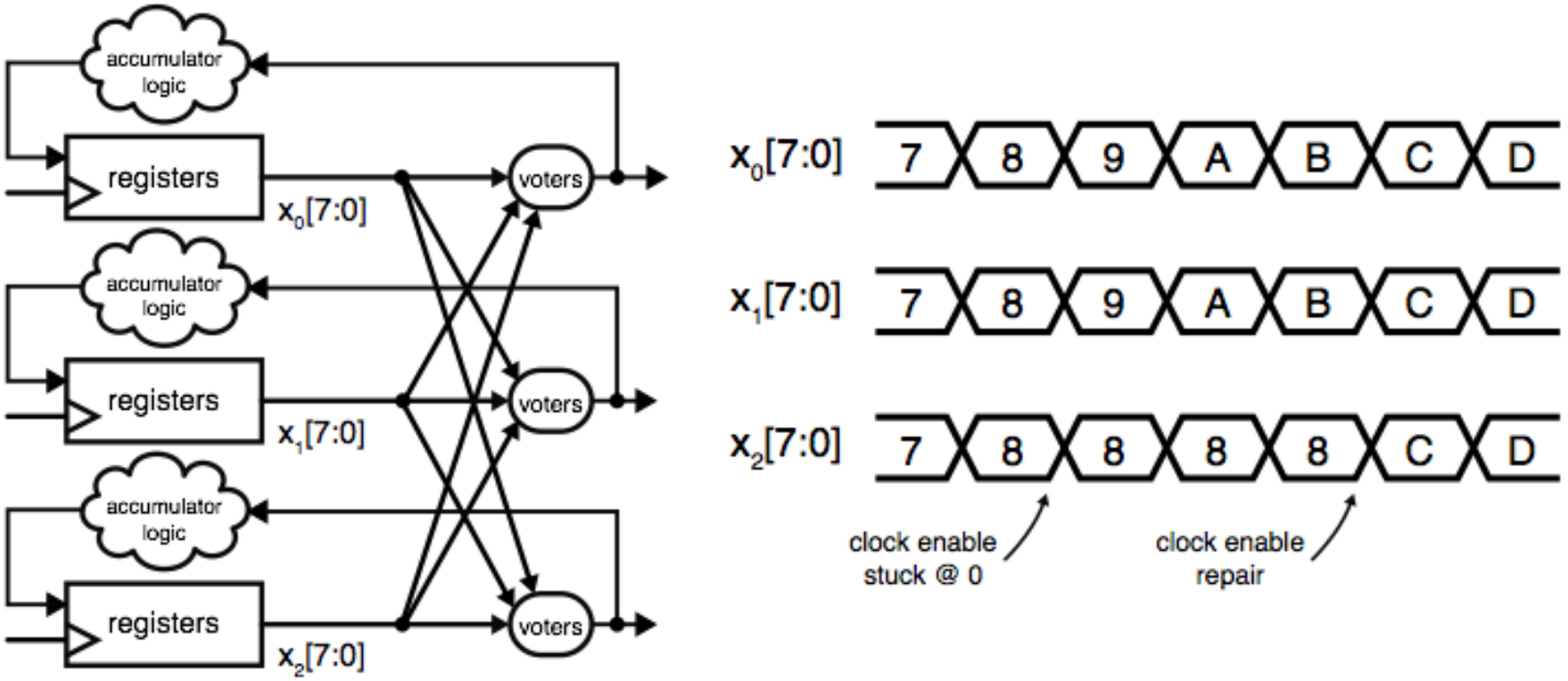
TMR Synchronization

- Fault repair through scrubbing
  - *Fixes the cause of the error*
  - *Does NOT fix the state of the circuit*
- State of circuit *must* be synchronized to working circuits



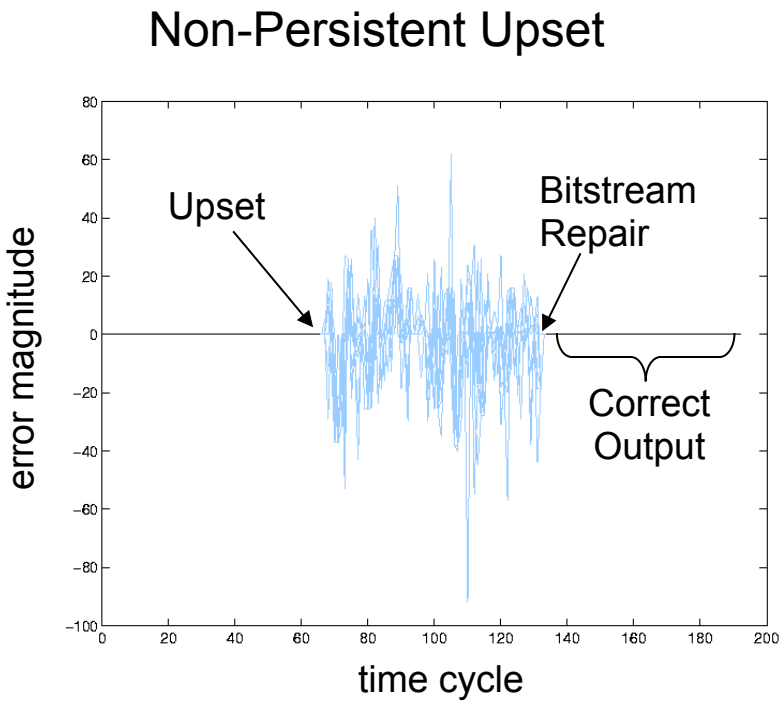


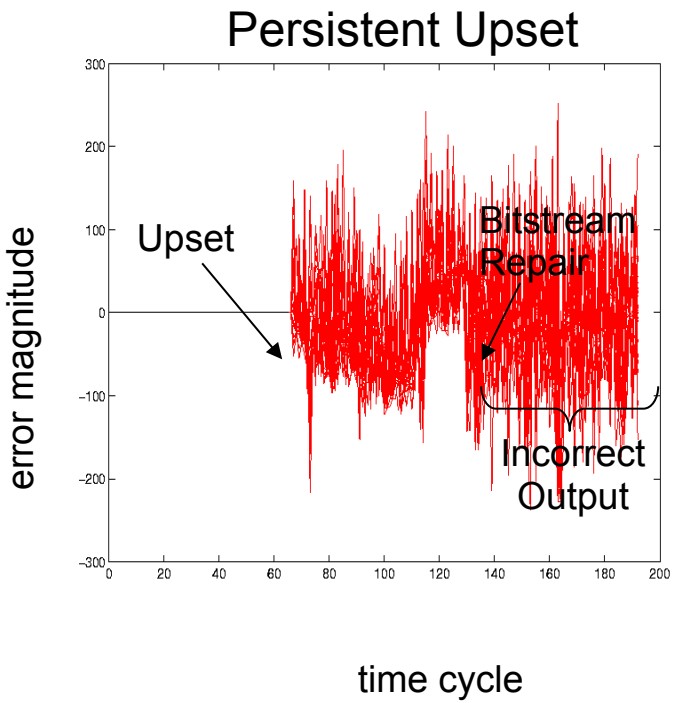
Synchronizing "Feedback" Voters



Persistent vs. Non-persistent Upsets

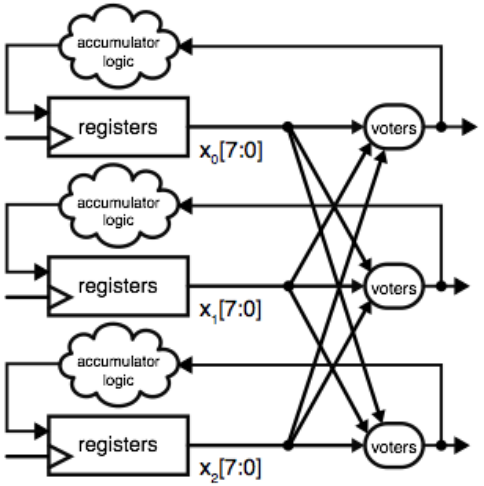
- Some upsets repaired through *scrubbing*
  - Non-persistent upsets: repairable through scrubbing
  - Persistent upsets: requires reconfiguration





Feedback TMR

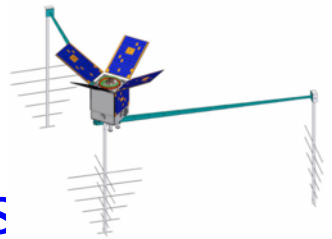
- "Cut" all circuit feedback with triplicated voters
  - *Identify feedback*
  - *Explore locations of voters*



- Advantages
  - *Provides self-synchronization*
  - *Frequent voting tolerates many MBUs*
- Disadvantages
  - *Voters in feedback loops reduce circuit timing*
  - *Can require significant resources*

BL-TMR

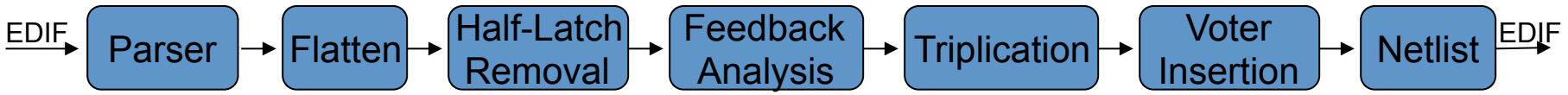
- BYU-LANL TMR Tool
  - *Developed at BYU under the support of Los Alamos National Laboratory (Cibola Flight Experiment)*
  - *Used to test TMR on many designs*
    - Fault injection, Radiation testing, in Orbit
  - *Testbed for experimenting with various TMR application techniques (used for research)*
- Source available online
  - *http://sourceforge.net/projects/byuediftools*
  - *Use/View at your own risk*



BL-TMR Software

- EDIF data structure & API
  - *Parse, represent, and manipulate EDIF*
- Available tools:
  - *EDIF parser*
  - *Half-latch removal*
  - *SRL replacement*
  - *Feedback cutset tool*
  - *Full and partial TMR*
  - *Detection circuitry insertion*
  - *EDIF output*
- Project size
  - *~50 Java packages*
  - *350+ Java classes*
  - *478,401 lines of code*
  - *Includes contributions from CHREC member LANL*

```
[brian@tiger:test] java -cp ~/jars/BLTmr.jar
byucc.edif.tools.tmr.FlattenTMR ../no_tmr/synth/counters80.edf --
removeHL --full_tmr --technology virtex -p xcv1000fg680 --log
counters80.log

BLTmr Tool version 0.2.3, 12 Oct 2006
Search for EDIF files in these directories: [.]
Parsing file ../no_tmr/synth/counters80.edf
Removing half-latches...
Flattening
            Flattened circuit contains 3451 primitives, 3461
nets, and 13692 net connections
Processing: ASUF 1.0

Forcing triplication of instance safeConstantCell_zero

Analyzing design . . .
            Full TMR requested.
Triplicating design . . .
domainreport=BLTmr_domain_report.txt
            Added 1931 voters.
            3431 instances out of 3451 cells triplicated (99% coverage)
            6862 new instances added to design.
            3431 nets triplicated (6862 new nets added).
            0 ports triplicated.
```



BL-TMR Versions

- Open Source
  - *Basic "Full TMR" tool for FPGA netlists*
  - *Provides user-driven TMR scripts*
  - *Limitations*
    - Supports Virtex and Virtex 4 devices
    - Not actively maintained, no support provided
    - Used primarily with Xilinx ISE (can be used with Vivado)
- NSF CHREC Version
  - *Updates supported by U.S. National Science Foundation Center for High-Performance Reconfigurable Computing (CHREC)*
  - *Support for 7 Series and Vivado Design Suite*
  - *Updated voter placement algorithms*
  - *Board Support packages (CSP, SpaceCube, etc.)*
  - *IP integration*

BL-TMR Design Steps

- Analyze and Merge Design
  - *Integrate IP and black boxes*
  - *Merge pre-TMR circuit IP*
- Low-Level Circuit Analysis
  - *IOB analysis and preparation*
  - *Clock tree and domain analysis*
- Feedback Analysis
- Voter Selection
- TMR Identification
- Netlist Generation

Sample Execution

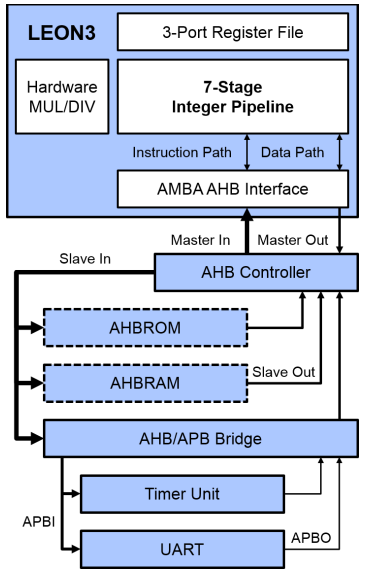
```
[brian@tiger:test] java -cp ~/jars/BLTmr.jar byucc.edif.tools.tmr.FlattenTMR ../no_tmr/
synth/counters80.edf --removeHL --full_tmr --technology virtex -p xcv1000fg680 --log
counters80.log

BLTmr Tool version 0.2.3, 12 Oct 2006
Search for EDIF files in these directories: [.]
Parsing file ../no_tmr/synth/counters80.edf
Removing half-latches...
Flattening
          Flattened circuit contains 3451 primitives, 3461 nets, and 13692 net
connections
Processing: ASUF 1.0

Forcing triplication of instance safeConstantCell_zero

Analyzing design . . .
          Full TMR requested.
Triplicating design . . .
domainreport=BLTmr_domain_report.txt
          Added 1931 voters.
          3431 instances out of 3451 cells triplicated (99% coverage)
          6862 new instances added to design.
          3431 nets triplicated (6862 new nets added).
          0 ports triplicated.
```

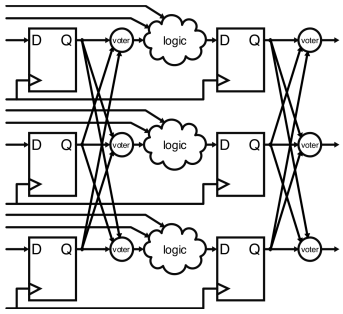
Cost of TMR

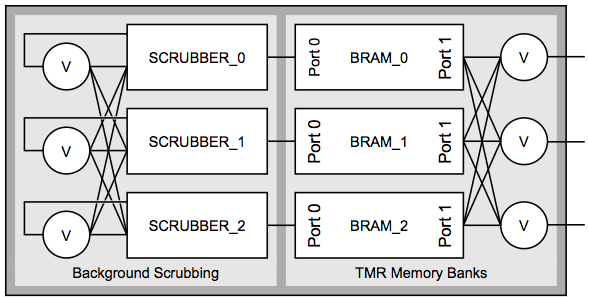
	Size Increase	Critical Path Before TMR	Critical Path After TMR	% Increase in Critical Path
blowfish	3.1X	28.3 ns	31.7 ns	12.0%
des3	3.4X	11.1 ns	13.6 ns	22.5%
qpsk	3.1X	80.0 ns	83.9 ns	4.9%
free6502	3.3X	29.6 ns	33.1 ns	11.8%
T80	3.3X	27.8 ns	33.7 ns	21.2%
macfir	3.9X	14.4 ns	19.5 ns	35.4%
serial_divide	4.1X	9.2 ns	12.2 ns	32.6%
planet	3.1X	10.9 ns	12.6 ns	15.6%
s1488	3.1X	9.9 ns	12.0 ns	21.2%
s1494	3.1X	10.4 ns	12.2 ns	17.3%
s298	3.1X	15.8 ns	19.1 ns	20.9%
tbk	3.9X	10.3 ns	12.9 ns	25.2%
synthetic	4.0X	9.9 ns	10.4 ns	5.1%
lfsrs	6.3X	9.0 ns	12.7 ns	41.1%
ssra_core	3.5X	6.1 ns	7.2 ns	18.0%
mean	**3.6X**	**8.17 ns**	**12.08 ns**	**16.0%**

TMR Experiment – LEON3

- How does TMR improve the reliability of the LEON3 operating on a Kintex 7 FGPA?
  - *Testing Core Architecture only*
  - *Excluded: Caches, Interrupt Controller, MMU, Debug Support Unit, Memory Controllers*
- Mitigation Approach
  - *Apply Feedback TMR on soft logic*
  - *Configuration scrubbing on FPGA*
  - *BRAM: TMR + memory scrubbing*





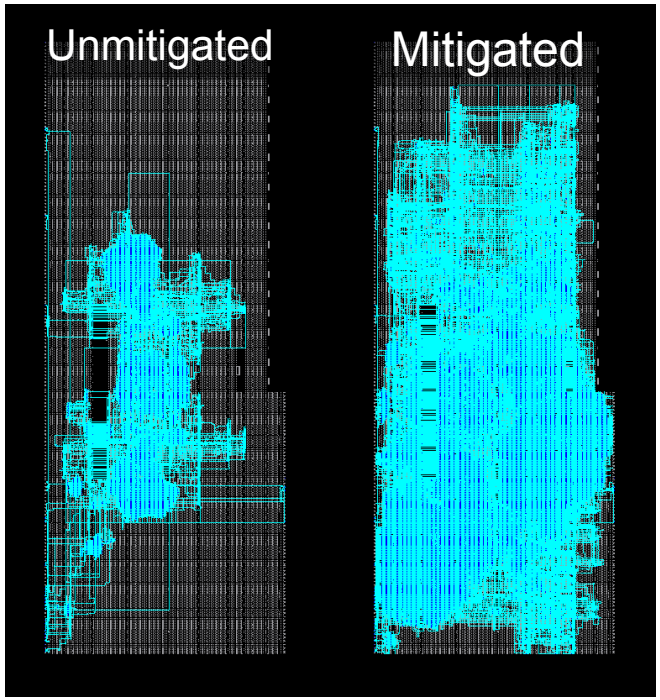


Michael Wirthlin, Andrew Keller, Chase McCloskey, Parker Ridd, David Lee, and Jeffrey Draper, "*SEU Mitigation and Validation of the LEON3 Soft Processor Using Triple Modular Redundancy for Space Processing*", 2016 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA 2016), February 2016.

Design Implementations

	NonTMR (Slices)	TMR (Slices)
Testing Overhead	1753	1960
Leon3 Slice Reg Core 1	1383	6567
Leon 3 Core 2	1410	6767
Total	4546	15294
Device	8.9%	30.0%

Table 1: Dual-LEON3 Design Utilization

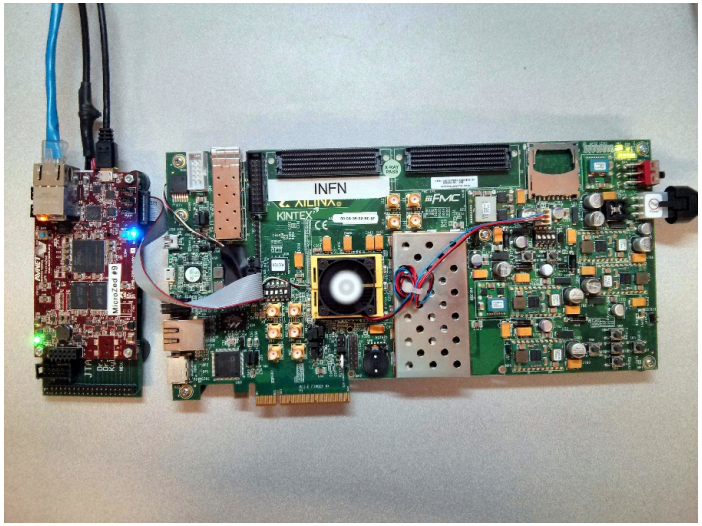


	NonTMR	TMR
No Detection	164 MHz (6.10 ns)	140 Mhz (7.15 ns)
Detection	132 MHz (7.56 ns)	73.7 MHz (13.56 ns)

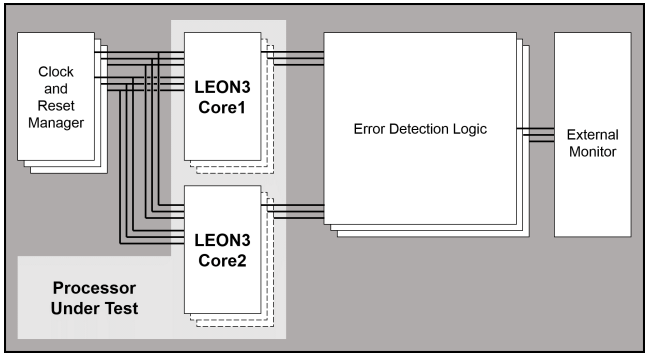
Table 2: Post-PAR Timing Summary

Fault Injection

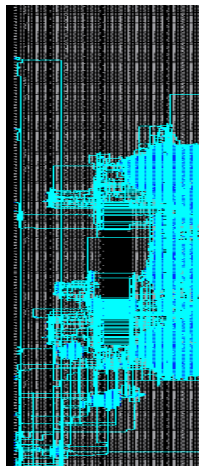
- Emulate configuration faults by injecting upsets through partial reconfiguration
  - *BYU JTAG Configuration Manager (JCM)*
  - *100 faults/second*
  - *Inject faults until an error is detected (Mean 'Upsets' to Failure)*



- Error Detection
  - *Instance two copies of LEON3*
  - *Triplicated detection circuitry*
- See demonstration



LEON3 Fault Injection Results

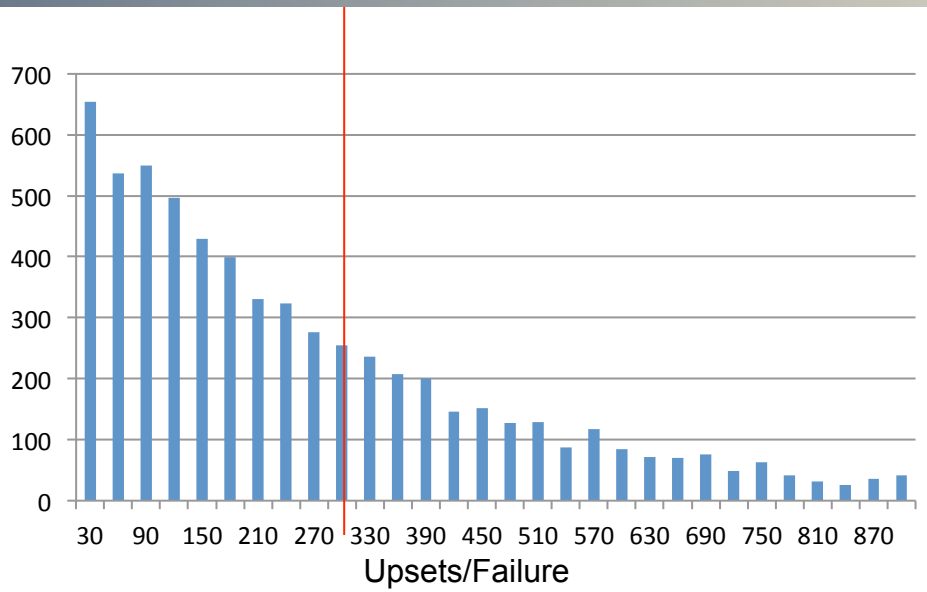


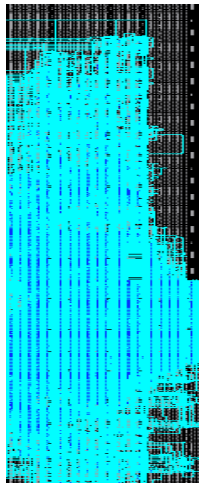
Non-TMR

mean upsets/failure = 282

runs = 6051

upsets = 1,831,859





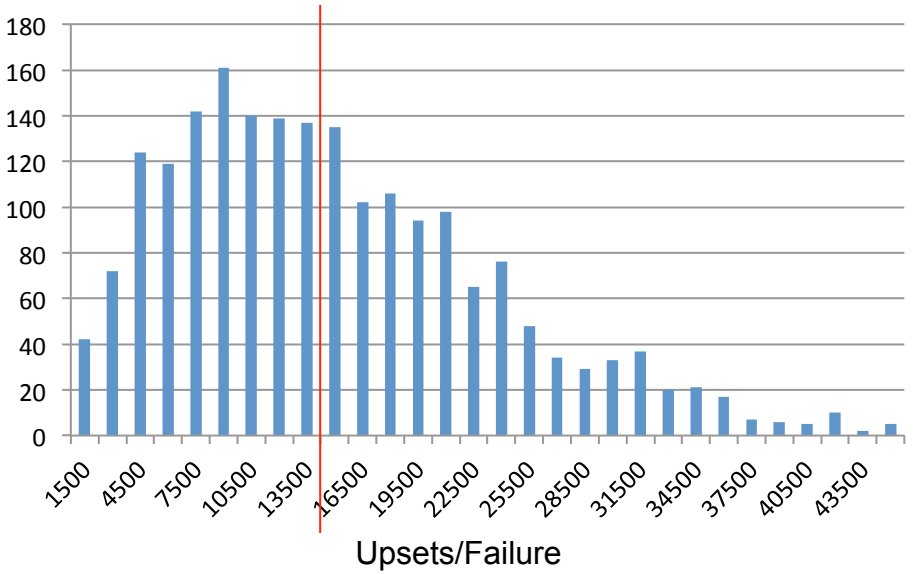
TMR

mean upsets/failure = 14,455

runs = 2037

upsets = 29,443,885

51x improvement



Fault Injection Results

Description	Unmitigated	TMR No Scrubbing	TMR BRAM Scrubbing	TMR FPGA Scrubbing	TMR Both Scrubbing
Faults Injected (n)	1,831,859	1,369,445	1,502,340	8,840,565	29,443,885
Observed Failures (k)	6,501	1,200	1,100	1,150	2,037
MUTF	282	1,141	1,366	7,687	14,455
Est. Sensitive Bits	240,539	59,393	49,627	8,817	4,689
Improvement	1.00×	4.05×	4.85×	27.28×	51.30×

FAULT INJECTION RESULTS

- Unmitigated: Original design with no mitigation
- TMR
  - *No scrubbing: BRAM and FPGA Faults accumulate*
  - *No FPGA Scrubbing/FPGA Scrubbing*
  - *FPGA scrubbing/No BRAM scrubbing*
  - *BRAM and Configuration scrubbing (no accumulation of errors)*

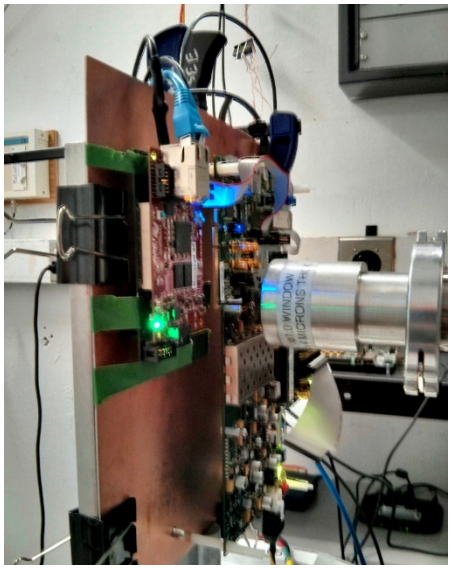
Heavy Ion Testing

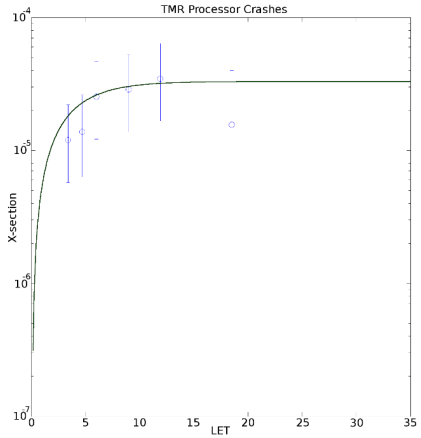
- Estimate orbital failure rate

LET (Ion) (MeV-cm^2/mg)		Mitigated (cm^2/proc.)	Unmitigated (cm^2/proc.)	Improv.
3.4	(Ne)	2.40E-5	3.65E-4	15.2
4.7	(Ne)	2.76E-5	2.58E-4	9.4
6	(Ne)	5.07E-5	5.38E-4	10.6
9	(Ne)	5.75E-5	5.59E-4	9.7
11.9	(Ar)	6.94E-5	5.43E-4	7.8
18.5	(Ar)	3.13E-5	2.69E-4	8.6

- Test Challenges
  - *Scrubbing problems*
  - *Global clocking issues*
- GEO Orbit Estimates

Design	Failure Rate (λ) (failures/processor/s)	MTTF (days/years)
Unmitigated	2.77E-8	501/1.4
Mitigated	4.15E-10	27,889/76





Neutron Testing

- Address challenges of heavy ion test
  - *Improved scrubbing hardware (full device)*
  - *Robust clocking*
- Neutron Test at Los Alamos Neutron Science Center (LANSCE)

Mode	1	2	3	4	5
Description	Unmitigated	TMR No Scrubbing	TMR BRAM Scrubbing	TMR FPGA Scrubbing	TMR Both Scrubbing
Failures	35	5	17	9	11
Fluence (n/cm^2)	1.34E+10	1.56E+10	1.06E+11	9.30E+10	2.06E+11
Fluence to Failure	3.83E+8	3.13E+9	6.24E+9	1.03E+10	1.87E+10
Cross Section (cm^2) (confidence int.)	2.61E-9 (1.73E-09, 3.49E-09)	3.20E-10 (3.38E-11, 6.07E-10)	1.60E-10 (8.26E-11, 2.38E-10)	9.68E-11 (3.23E-11, 1.61E-10)	5.34E-11 (2.12E-11, 8.56E-11)
Improvement	1.00×	8.16×	16.27×	26.94×	48.85×
Fault Injection	1.00×	4.05×	4.85×	27.28×	51.30×

Single Point Failures (SPF)

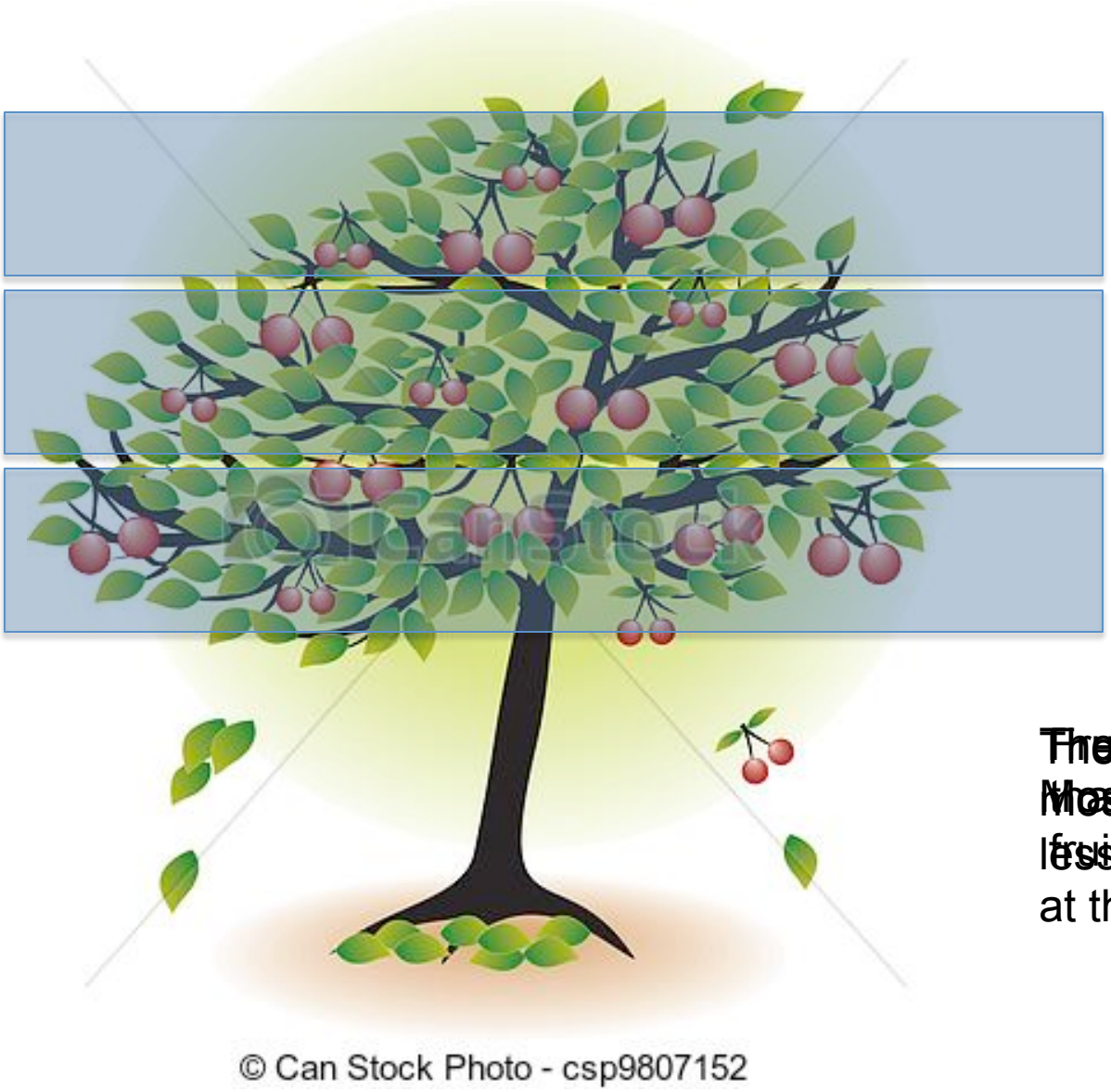
- Netlist-level feedback TMR did not remove all sensitive configuration bits
  - *Estimated remaining Sensitive bits: 4,700*
  - *Each bit is a "single-point failure" (SPF)*
- Source of SPFs
  - *Constants shared with TMR domains*
    - Vivado tools combine constants
  - *Placement/Routing TMR Domain conflicts*
    - Routing Shorts/Shared Mode (VERI-Place tool)
  - *Design Single-point failures*
    - Clocks, I/O, JTAG/BSCAN

Low Hanging Fruit

A child picking fruit, Gerard van Honthorst
Het Loo Palace, Apeldoorn

"a course of action that can be undertaken quickly and easily as part of a wider range of changes or solutions to a problem"

Low Hanging Fruit



High Fruit

Middle Fruit

Low Hanging Fruit

The highest fruit requires the most effort to pick. There is less fruit at the top of the tree than at the bottom and the middle.

Harvesting the SEU "Fruit"

"Fruit" – Sensitive configuration bits in an FPGA design. We want to 'pick' as many of them as possible.

"Picking" the configuration bits involves mitigating the design so these configuration bits no longer cause design errors.

It is more and more difficult to pick the "fruit" as it is higher in the tree.

The amount of "fruit" to pick depends on the amount of effort you are willing to invest in the harvest.

It may not be worth it to get "all" of the fruit out of the tree.

LEON3 "Fruit"

LEON3 Processor: 240,000 sensitive bits

Top Fruit: Unknown
~200 (<.1%) - 0
∞ improvement

High Fruit: Design SPF
~500 (.2%) - 200
1200x improvement

Middle Fruit: TMR Placement/Routing
~1,500 (.6%) - 700
343x improvement

Middle Fruit: Constant Routing
~3,000 (1.3%) – 1,700
141x improvement

Low Fruit: TMR (netlist) + Scrubbing
54,700 (23%) - 4,700
51x improvement

Low Fruit: TMR (netlist)
180,600 (75%) – 59,400
4x improvement

Technique	Mitigated	Sensitive Bits	Improvement
Unmitigated	0	240,000	1x
TMR (netlist)	180,600 (75%)	59,400	4x
TMR+Scrubbing	54,700 (23%)	4,700	51x
Constant Trees	3,000 (1.3%)	1,700	141x
Placement/ Routing	1,500 (.6%)	700	343x
Design SPF	500	200	1200x
Unknown	200	0	∞

TMR Going Forward

- Low-level TMR enhancements
  - *Unique constant generation*
  - *Multi-domain routing conflicts (post-routing TMR)*
- Investigation into complex designs/structures
  - *Soft-Processor Cores*
  - *Multi-core SOCs*
- Integration of other mitigation approaches
- Improved timing aware TMR
- Verification support
- GUI support
- Enhanced voting options and automatic selection

Summary

- TMR is effective at mitigating SEUs for SRAM-based FPGAs
  - *Must be coupled with configuration scrubbing*
- Feedback TMR provides self-synchronization
- The BL-TMR tool has been used to mitigate many FPGA designs (LEON3 Soft processor)
  - *Fault Injection Results*
  - *Radiation Testing*
- TMR is not sufficient for mitigating all SEUs
  - *Memory ECC/Memory scrubbing*
  - *Additional placement/routing aware tools needed*
- TMR is an important "low hanging fruit" approach to SEU mitigation

Questions?