Characterizing the Vulnerability of an Onboard Computation to Silent Data Corruption

Kenneth M. Zick

University of Michigan
NASA LaRC GSRP Fellowship

John P. Hayes

University of Michigan

Outline

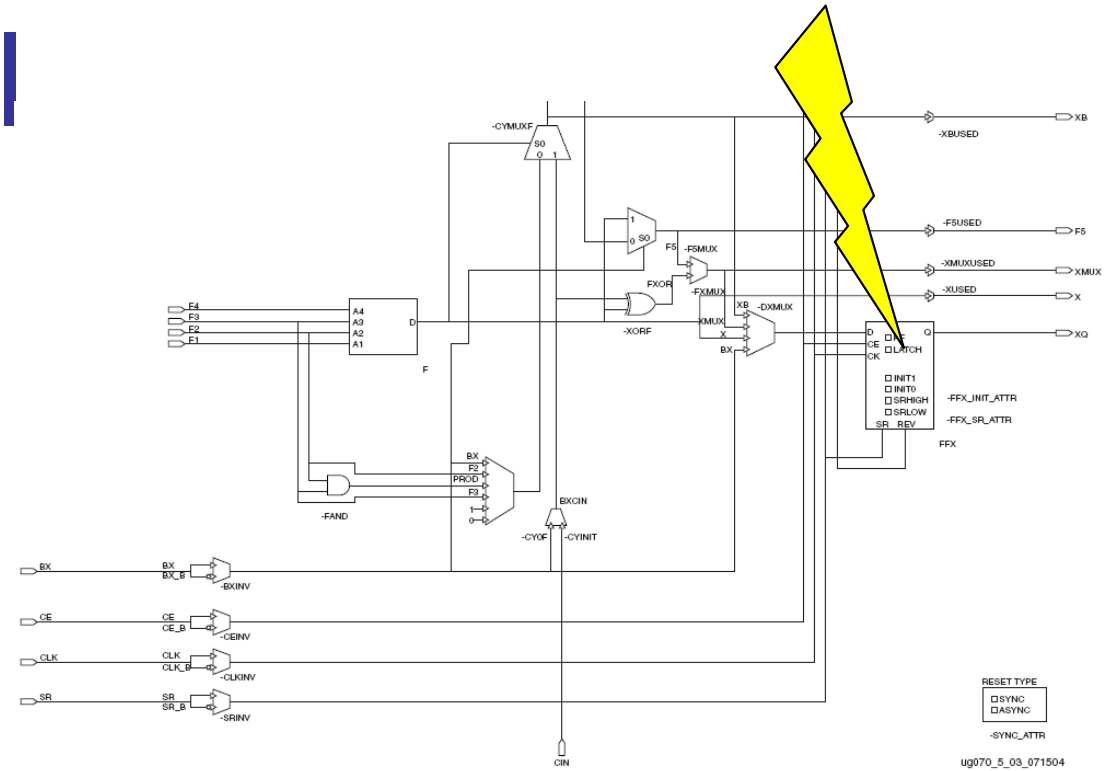
- Introduction
- Vulnerability metric
- Experimental results
- Conclusions

Motivation

- Goal: characterize the vulnerability of a particular computation to silent data corruption
- Need to understand vulnerability so we can:
  - Decide whether a non-redundant implementation is acceptable
  - Compare alternative implementations of an algorithm
  - Decide which regions of a computation deserve soft error protection

Fault model

- Single event upset (SEU) in a user logic flip-flop:



Portion of a Virtex-4 SLICEL, from Virtex-4 FPGA User Guide

- Our study explicitly does not include:
  - SEUs in memories or configuration bits (assumed to be detectable)
  - SEUs in DSP blocks
  - Single event transients

Error model

- Our focus is on “insidious soft errors” (ISEs)
- ISEs are logic circuit errors that:
  - Escape error detection
  - Cause a corrupted computational result
- Relation between ISEs and silent data corruption (SDC):
  - ISEs cause SDC, but there are many other causes of SDC as well
  - ISEs can start long before the SDC occurs, due to error latency

Which metric to use?

- Goal: characterize the threat of an insidious soft error
  - Need to measure the extent of the vulnerability per computation
  - Need a measure during the design phase; SEU rate may be unknown
- One approach: Soft error rate (SER), MTTF
  - Requires knowledge of SEU rate
  - Does not account for throughput
    - Ex.: Impl. A has SER=1 FIT and throughput of 1 results/s, while impl. B has SER=2 FIT and 5 results/s. Results from B are more reliable despite higher SER.
- Another approach: measure the fraction of faults that lead to errors
  - Names: architectural vulnerability factor (AVF), logic derating, error cross-section
    - *vulnerability fraction* (VF)
  - Does not capture the absolute extent of the vulnerability
    - For that we need to integrate across space & time

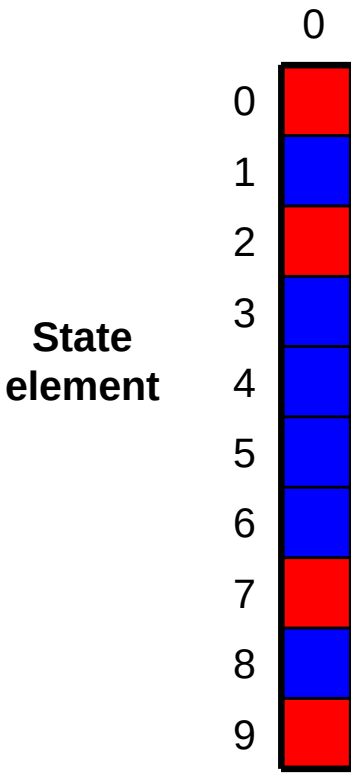
Vulnerability over Space & Time (*VST*)

- *N* state elements with spatial weight $w_s(n)$
- *K* temporal intervals with weight $w_t(k)$
- Vulnerability at a particular space & time: $0 \leq v(n,k) \leq 1$
- *VST* is a straightforward summation of vulnerabilities per computation:

$$VST = \sum_{n=1}^{N} \sum_{k=1}^{K} v(n,k) \times w_s(n) \times w_t(k)$$

- Units: (spatial elements) · (time), e.g. bit·sec
  - *VST* represents the intrinsic vul. of a computation, independent of SEU rate
    - Similar metrics: mean-work-to-failure[4], data vulnerability of an application[5] (in MB-seconds)

VST map



Assuming all elements consist of 1 bit and all intervals are 1 s:

***VST* = 36 bit·sec**

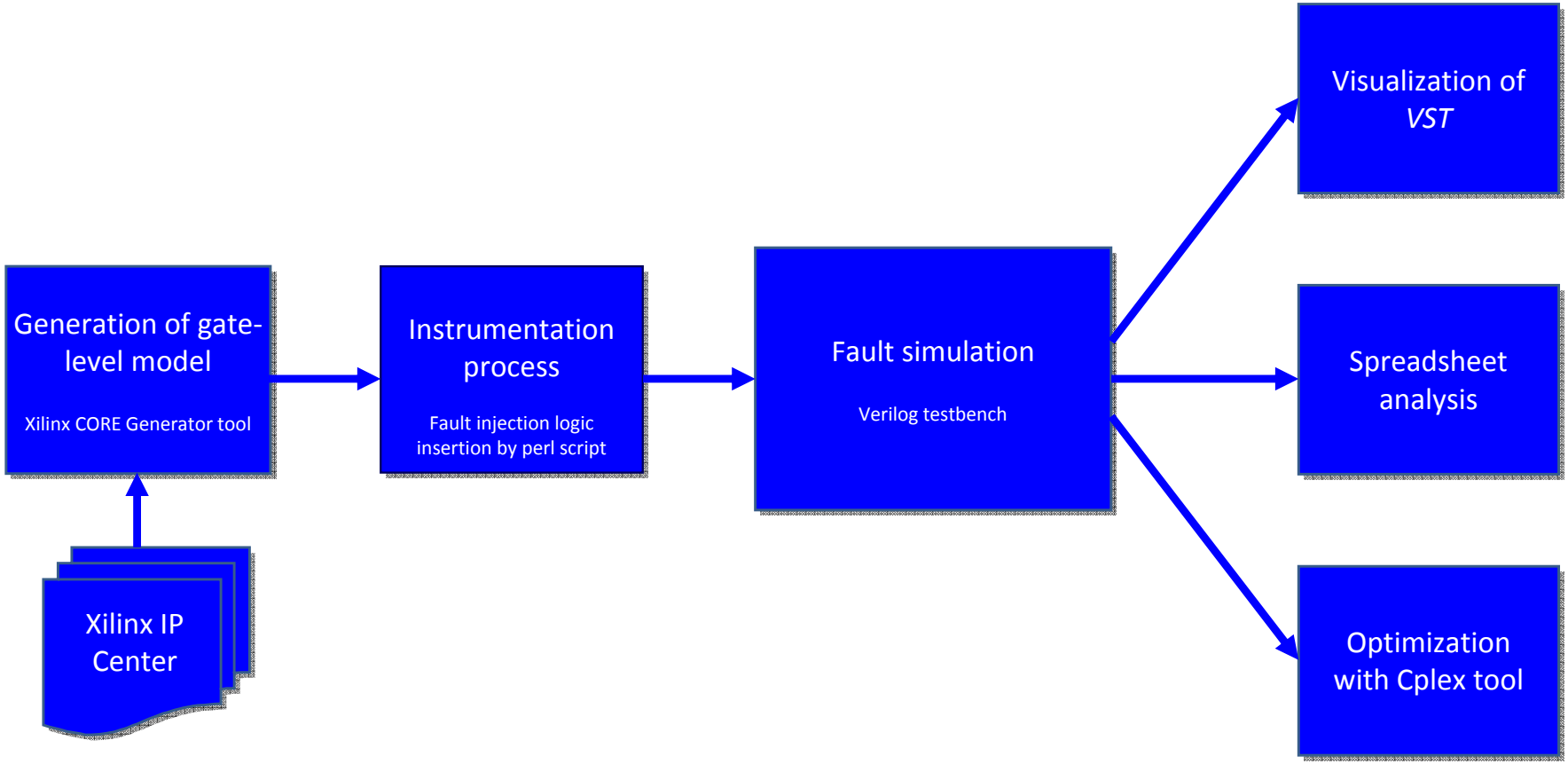
If upset rate is uniform and assuming a single fault per computation: $P_{err} = VST \times \lambda$

Example: λ = .01 FIT

$P_{err} = (36 \text{ bit·sec})(2.8 \times 10^{-15}/\text{bit·sec})$

$P_{err} = 10^{-13}$

Methodology Flow



Experiment 1

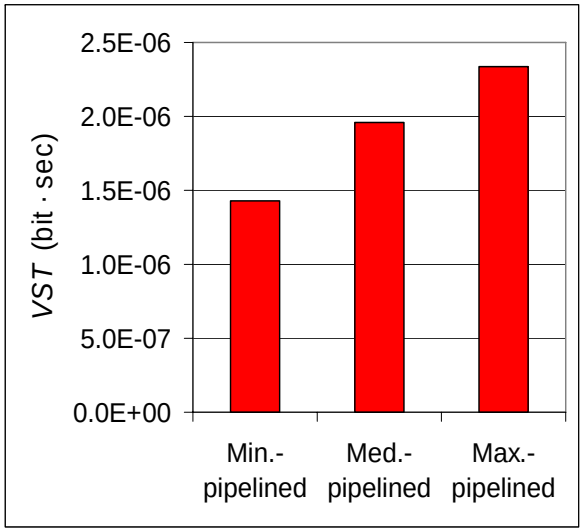
- Question: How does *VST* vary with different amounts of pipelining?
- Generated 3 implementations of a double-precision floating-point adder, based on the Xilinx Floating Point Operator v3.0[6]. Target was Virtex-5. Implementation characteristics:

Implementation	No. of pipeline stages	No. of state elements (flip-flops)	Clock period (ns)	Throughput (FLOPs/s)
Min.-pipelined	2	147	10.31	97M
Med.-pipelined	7	732	4.02	249M
Max.-pipelined	12	1085	3.16	316M

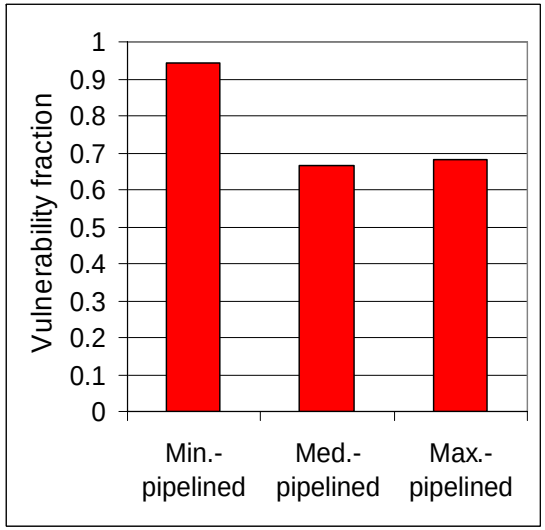
- Performed exhaustive fault simulation for 10 random FP addition operations

Results

Proposed *VST* metric



Existing VF metric



Conclusions:

- Minimum-pipelined implementation has lowest vulnerability (*VST*)
- Adding pipe stages (in this case) increases frequency and throughput but adds vulnerable elements; net increase in *VST*
- Existing VF metric gives incomplete picture

Experiment 2

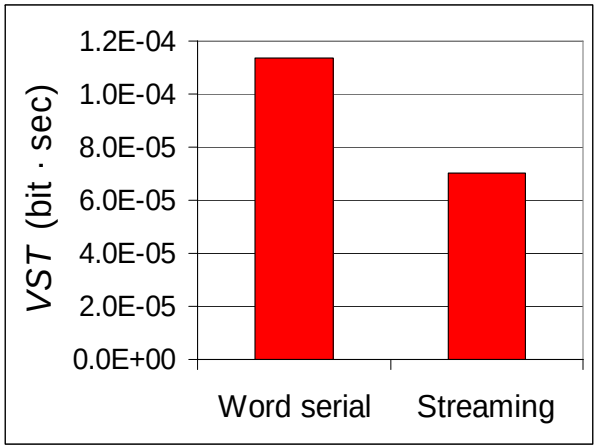
- *VST* for a serial vs. streaming implementation
- Generated two implementations of the COordinate Rotation DIgital Computer (CORDIC) algorithm, based on the Xilinx CORDIC v3.0[7]. Target was Virtex-II Pro. Implementation characteristics:

Implementation	No. of state elements (flip-flops)	Clock period (ns)	Throughput (ops/s)
Word serial	535	7.52	7M
Streaming	1341	5.88	170M

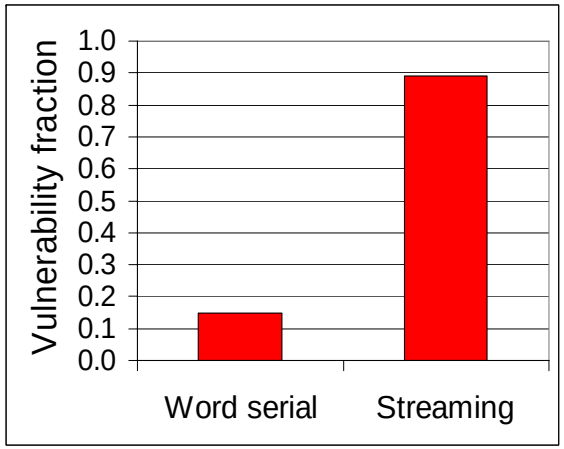
- Performed exhaustive fault simulation for ten random vector rotation operations

Results

Proposed *VST* metric



Existing VF metric



Conclusions:

- The streaming implementation is the least vulnerable (lowest *VST*)
- In this case the highest performing implementation also has lowest vulnerability, whereas the reverse was true in Experiment 1.
- Again the existing VF metric gives a (very) incomplete picture

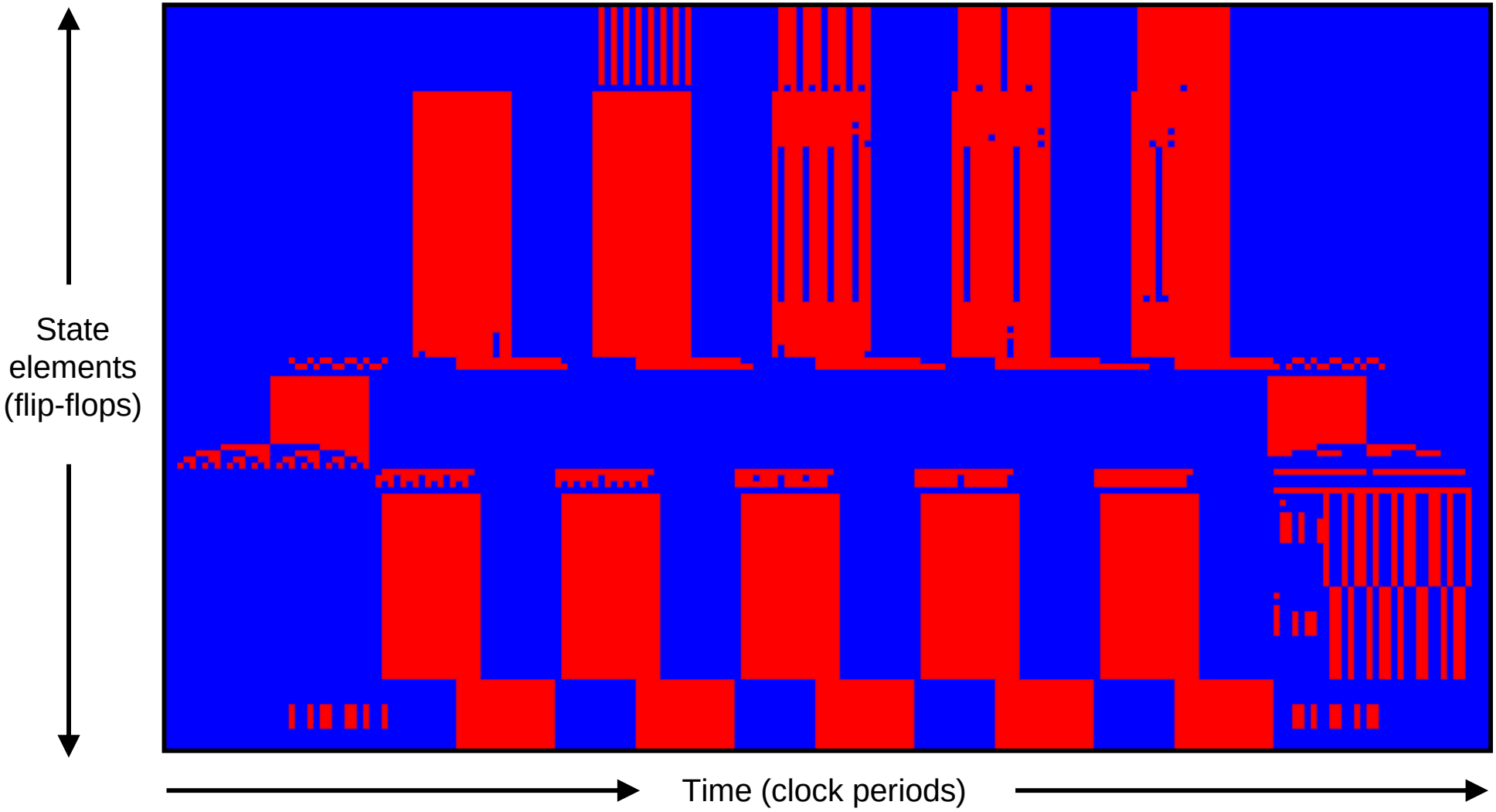
Experiment 3

- *VST* across different FFT implementations
- Generated three implementations of a 1D complex fixed-point Fast Fourier Transform, all based on the Xilinx FFT v5.0[8]. Target was Virtex-II Pro.

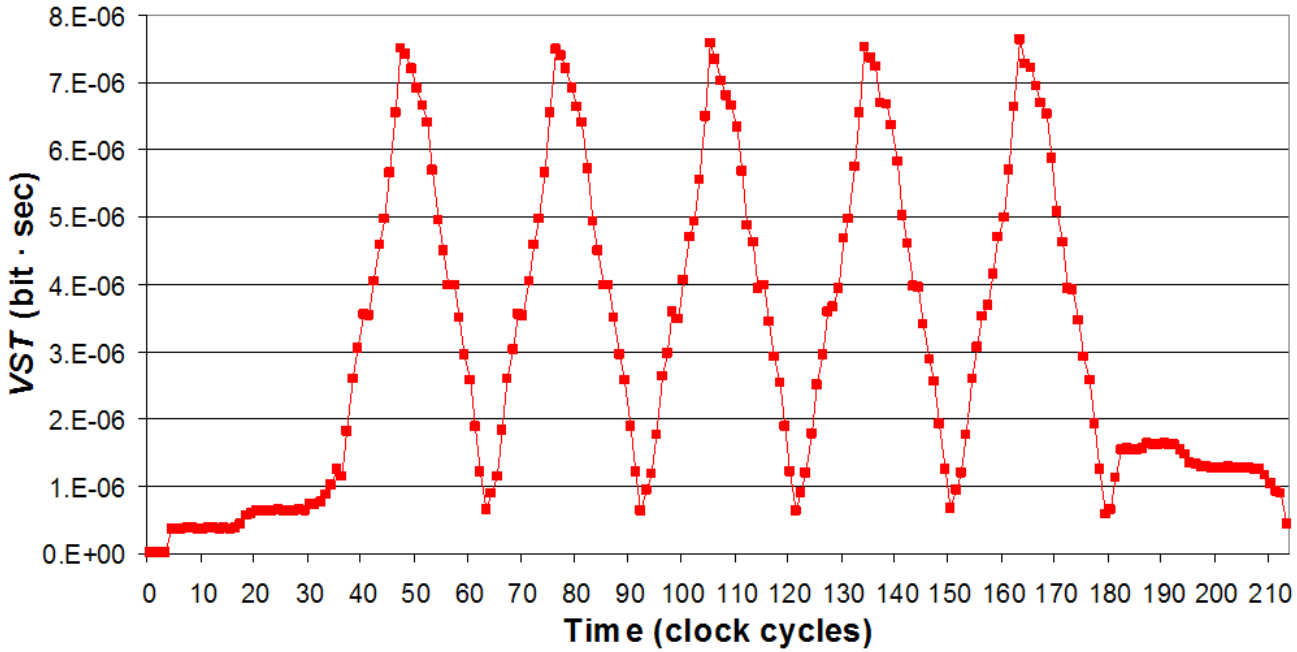
Implementation	No. of state elements (flip-flops)	Clock period (ns)	No. of engines	Length of each compute phase (clock cycles)
Radix-2 Lite	1459	8.77	1	$(\log_2 N) + 12$
Radix-2	1566	6.58	2	$(0.5 \log_2 N) + 13$
Radix-4	3715	6.45	4	$(0.25 \log_4 N)+16$

- Performed fault simulation of 32-pt and 64-pt FFTs
- Built a simple model of *VST* vs. FFT size for each implementation, allowing *VST* to be estimated

VST map: 32-pt Radix-2 FFT



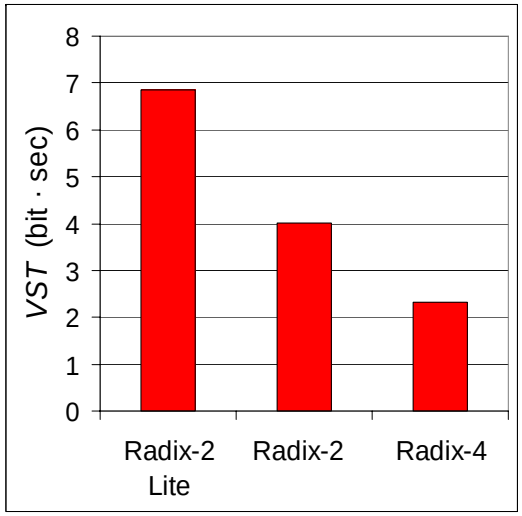
Vulnerability dynamics



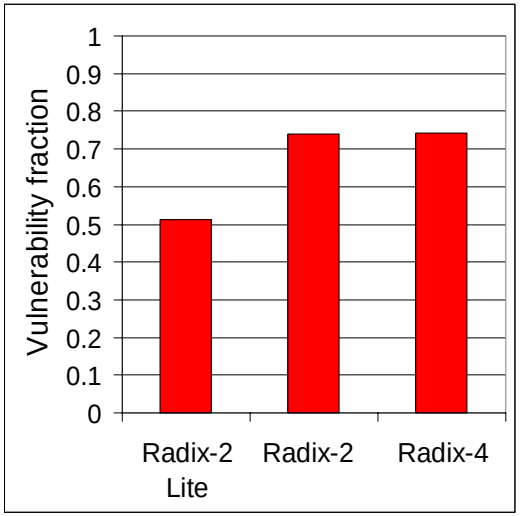
Dynamical behavior of *VST* for 32-point Radix-2 FFT

Results

Proposed *VST* metric



Existing VF metric



Projected vulnerability for 64K FFT

Conclusions:

- This time, the highest performing implementation also has lowest vulnerability (*VST*)
- Again the existing VF metric gives a (very) incomplete picture

Conclusions

- Proposed approach allows:
  - the intrinsic vulnerability of a specific computation to be characterized
  - alternative implementations to be compared w/ a single figure-of-merit
  - spatial and temporal patterns to be uncovered
  - challenge: extensive fault injection usually required
- Vulnerability experiments indicate:
  - Inefficient implementations have low VF, but can also have high vul.
  - *VST* can be treated as an independent design criterion along with power consumption, area, performance

Acknowledgments

- NASA Langley Research Center GSRP Fellowship
- Kevin Somervill & Electronics Systems Branch, NASA Langley Research Center
- John Holland, University of Michigan

Thank you!

References

1. Kenneth M. Zick and John P. Hayes, “High-Level Vulnerability over Space and Time to Insidious Soft Errors”, *IEEE International High-Level Design, Validation and Test Workshop (HDLVT 2008)*, November 2008 (to appear)
2. J. Greco, G. Cieslewski, A. Jacobs, I. Troxel and A. George, "Hardware/Software Interface for High-Performance Space Computing with FPGA Coprocessors," *2006 IEEE Aerospace Conference*, May 2006.
3. S. Mukherjee, *Architecture Design for Soft Errors*, Morgan Kauffman, Burlington, MA, 2008.
4. G. Reis, J. Chang, N. Vachharajani, R. Rangan, D. August and S.S. Mukherjee, “Design and Evaluation of Hybrid Fault-Detection Systems”, *International Symposium on Computer Architecture (ISCA)*, June 2005, pp. 148-159.
5. P. Springer, "Assessing Application Vulnerability to Radiation-Induced SEUs in Memory," NASA Technical Reports Server, 2001.
6. Xilinx Floating-Point Operator v3.0 Product Specification, http://www.xilinx.com/support/documentation/ip_documentation/floating_point_ds335.pdf. Accessed April 2008.
7. Xilinx CORDIC v3.0 Product Specification, http://www.xilinx.com/support/documentation/ip_documentation/cordic.pdf. Accessed April 2008.
8. Xilinx Fast Fourier Transform v5.0 Product Specification, http://www.xilinx.com/support/documentation/ip_documentation/xfft.pdf. Accessed April 2008.