SPFFI: Simple, Portable FPGA Fault Injector

CHREC
NSF Center for High-Performance Reconfigurable Computing

MAPLD 2009

UF UNIVERSITY of FLORIDA
Virginia Tech
VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY
BYU
BRIGHAM YOUNG UNIVERSITY
THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON DC

Grzegorz Cieslewski
Ph.D. Student
NSF CHREC Center, University of Florida

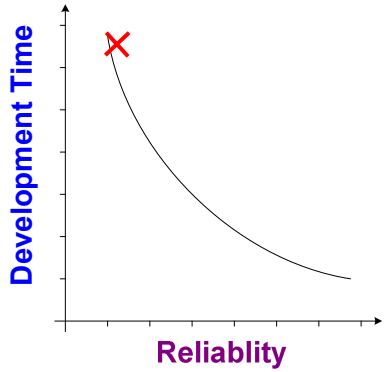
Dr. Alan D. George
Professor of ECE
NSF CHREC Center, University of Florida

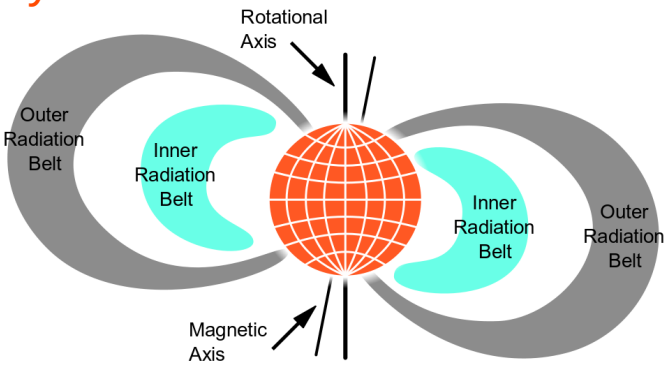
Outline

- Goals and Motivations
- Existing FPGA Fault-Injection Technology
- SPFFI Architecture and Capabilities
- Methodology of Fault Injection
- Fault Injection and Statistics
- Case Studies
  - MicroBlaze – System-Level Testing
  - LIDAR – Module-Level Testing
- Conclusions and Future Work

Goals and Motivations

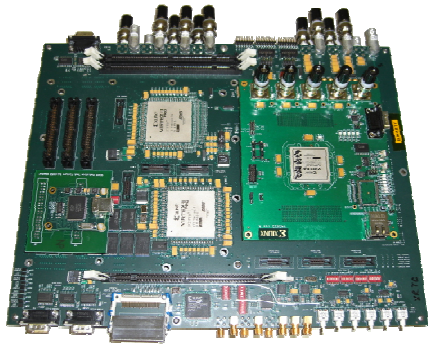
- **Goals –** Explore methodologies and procedures for effectively combining fault-testing methods and concepts
  - Verification of fault-tolerant FPGA architectures and designs for space and terrestrial applications
  - Portable solution that can be used on COTS as well as specialized FPGA platforms
- **Motivations –** FT testing of large FPGA-based designs can be very difficult
  - Why wait for fault testing until completion of system?
  - Beam testing is very expensive and lacks coverage
  - Developers need to be able to perform testing without specialized hardware to estimate reliability
  - Enabling technology for further research
- **Challenges –** Tool limitations and shortcomings of FPGA architecture
  - Restricted access to dynamic components (BRAM)
  - Slow programming interconnect

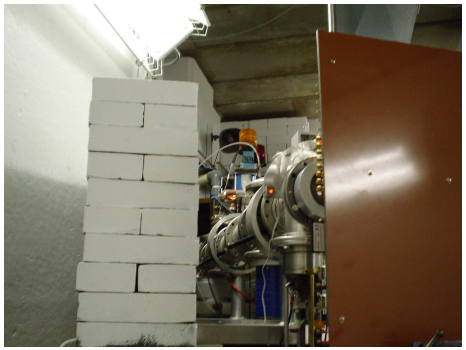




Existing FPGA Fault-Injection Tech.

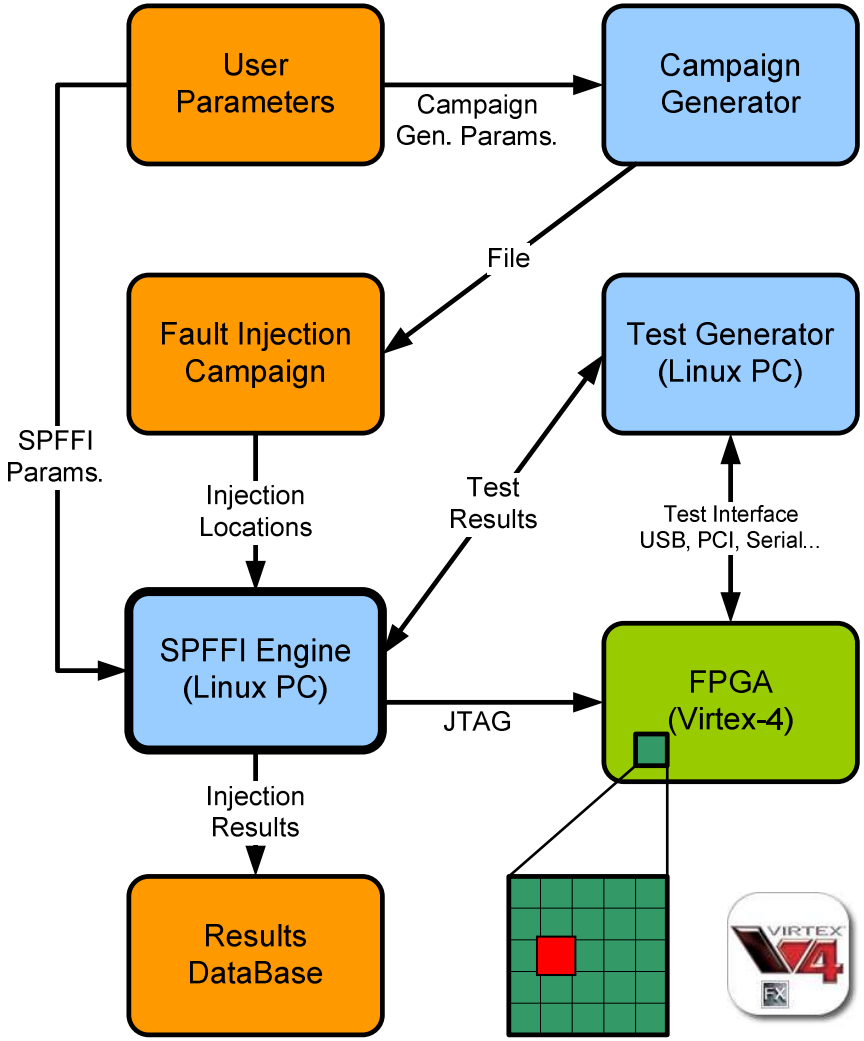
- Custom hardware solutions
  - XRTC fault-injection board
  - High injection speed and excellent coverage
  - Limited portability and difficult in-system testing
- Beam testing
  - Most accurately resembles space conditions
  - Very costly
    - High time and financial overhead
  - Few facilities which provide the service
  - Limited experiment repeatability
- Manual injection
  - Requires designer to manually insert faults in HDL
    - Can interfere with design at hand
    - Difficult to implement
  - Cannot accurately emulate types of faults expected





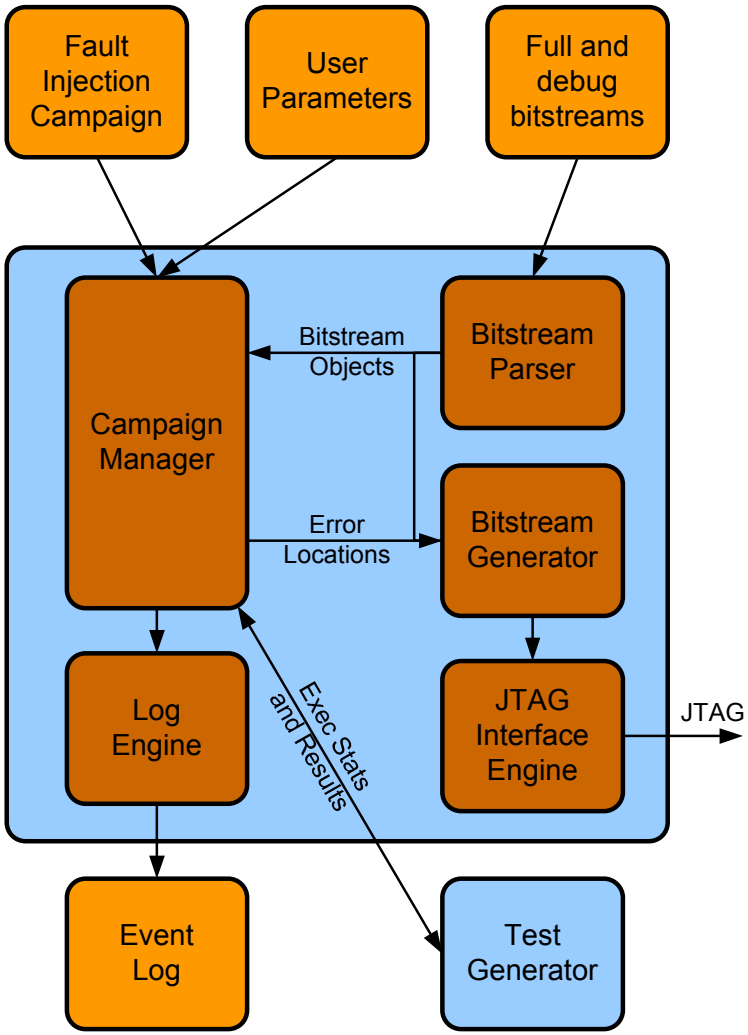
SPFFI – Simple Portable FPGA Fault Injector

- Introduction
  - New fault-injection utility designed for easy and portable use
  - Supports multiple Virtex-4 FPGA platforms with Virtex-5 and -6 support coming in near future
- System Model
  - PC connected to FPGA-based sys.
    - Prog. Interface: JTAG (most popular)
    - Test interface: USB, PCI, Serial, …
- SPFFI consists of 3 major components
  - SPFFI Engine
    - Responsible for fault injection and result collection
  - Campaign Generator
    - Crafts injection campaigns based on parameters specified by user
  - Test Generator
    - Plug-in component for customizable FPGA testing



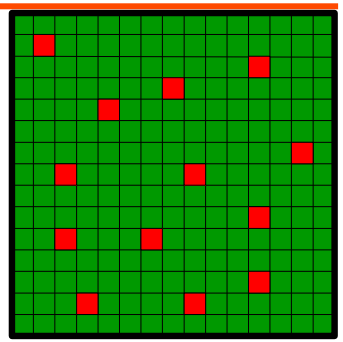
SPFFI Components

- SPFFI Engine
  - Campaign Manager
    - Orchestrates execution of campaign as specified in input file
  - Bitstream parser
    - Analyses input bitstreams to extract relevant architectural and design information
  - Bitstream generator
    - Crafts customized full and partial bitstreams to allow for fault injection and removal
  - JTAG interface engine
    - Provides high-level programming interface for variety of JTAG connections
  - Logging Engine
    - Stores events and injection results in a database
- Test Generator
  - Plug-in application that verifies correct operation of design
  - User-defined for maximum flexibility
  - Can be partially hosted on FPGA to speed up testing

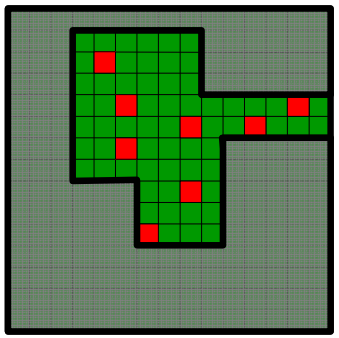


SPFFI Components

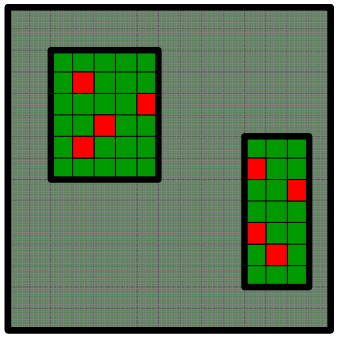
- Campaign Generator
  - Generates locations at which faults will be injected by SPFFI Engine
    - Parameters: injection count, resource type, transition type, occupied vs. unoccupied frames, campaign type
  - Campaign Types
    - Uniform campaign
      - Selects random injection locations anywhere on chip
    - Automatically targeted campaign
      - Based on coarse bitstream analysis; chip is divided into two mutually exclusive occupied and unoccupied regions
      - Two campaigns are performed, one for each region
      - Vast majority of observable errors are due to faults injected into occupied region
    - Manually targeted campaign
      - Region of interest specified by user



Uniform Campaign



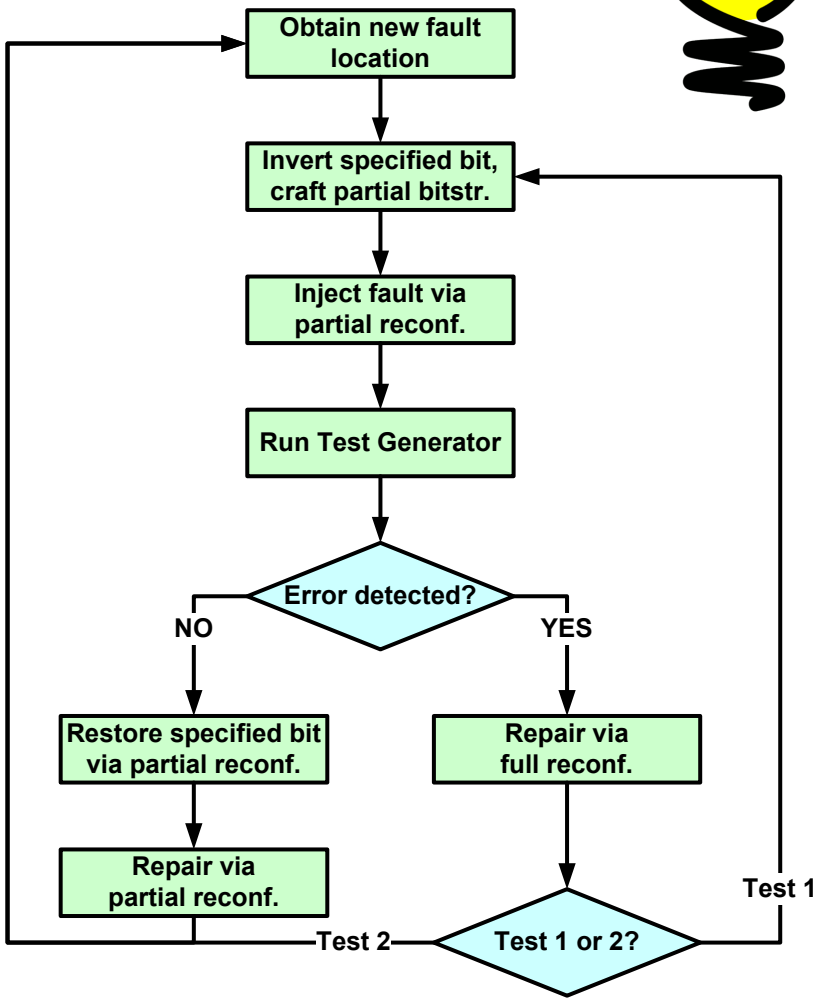
Automatically targeted



Manually targeted

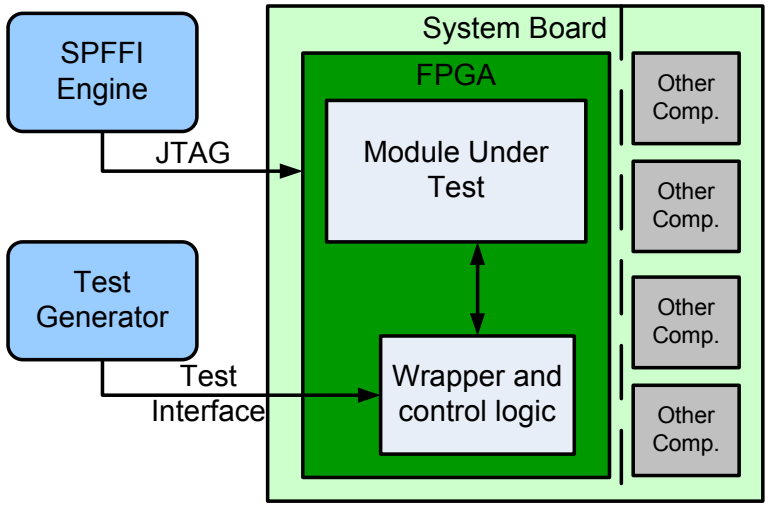
Fault Injection Methodology

- Fault-injection considerations
  - Correctness
    - Minimize false positives
    - Representative of how SEUs and SEFIs occur
  - Performance
    - Higher performance will provide better estimate of error rate
- Test Generator upshots
  - No observable error (benign fault)
    - Triggers fault removal via partial reconfiguration
  - Injected fault produces error
    - FPGA produces invalid or no output
    - Data error (comparison with golden standard fails)
    - Triggers return to original state via full reconfiguration
    - Site is re-tested to eliminate bias introduced by partial reconfigurations

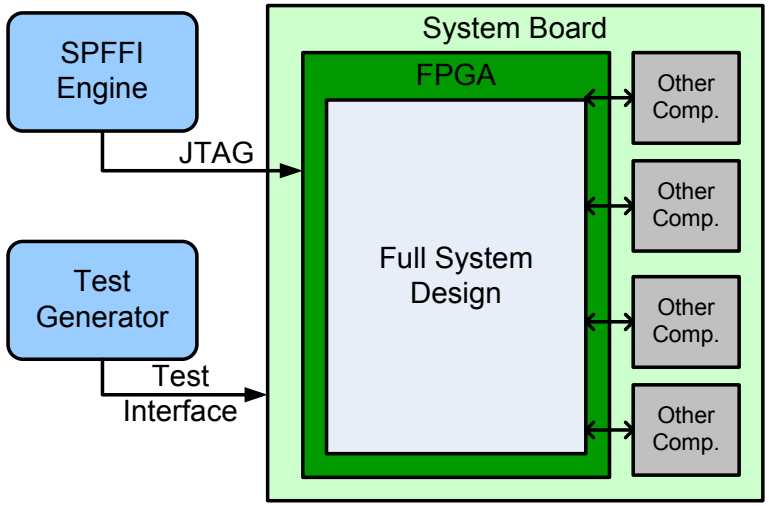


Testing Methodology

- System Types
  - Module-Level Testing
    - Designed to test standalone modules of a system
    - Data is provided and collected from module by a wrapper design residing on same chip
    - Test data can be provided by Test Generator or can reside on FPGA to increase testing speed.
  - System-Level Testing
    - Used for testing of systems which interact with external hardware
      - System-on-chip type scenarios
      - System with FPGA co-processor
    - Test Generator is used to issue commands for starting and stopping testing
  - Hybrid Testing
    - Combination of above approaches
    - Used for integration & incremental testing



Module-Level Testing

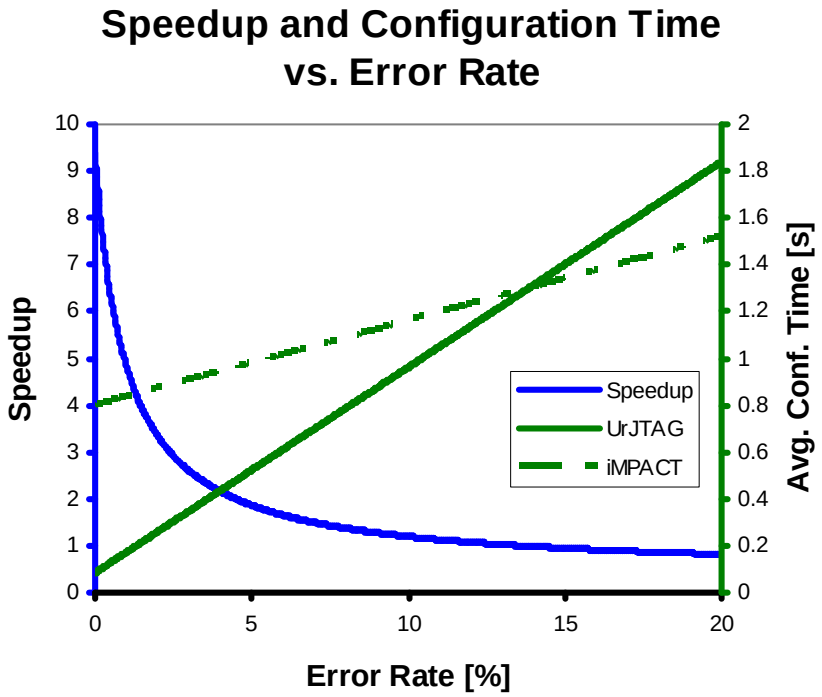


System-Level Testing

SPFFI Performance

- JTAG performance
  - Strongly dependent upon JTAG Engine backend
    - Up to 2 injections/s when using iMPACT
    - Up to 12 injections/s when using UrJTAG
    - Based on Virtex-4 LX25
  - Different backends possible
  - Limited by poor performance of JTAG cables and software overhead
- Test Generator performance
  - Generating representative set of test vectors is difficult
    - Representative set might be very large and require long testing times.
  - Varies depending on implementation
    - Physical location of test vectors
- Error rate
  - Less than 10% for most of designs
  - Injection speed generally increases for designs with lower error rates
    - Fewer full reconfigurations to restore known state

Programming Type	iMPACT	UrJTAG
Full Configuration	~4.0	8.8
Partial Reconfiguration	0.4	0.043



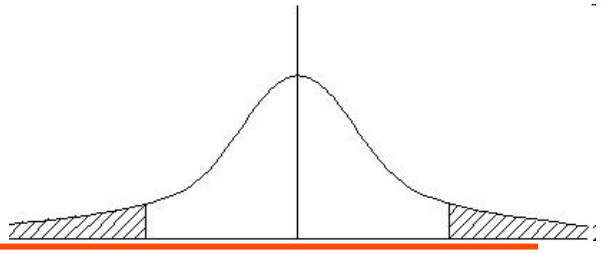
Fault Injection and Statistics

- Designing fault injection experiments with statistic in mind
  - Coverage of sensitivity testing
    - Full testing vs. partial testing
  - Select a representative bits in region of interest
    - Random sampling
    - Stratified sampling
  - Interpretation of results
    - Are results obtained representative of whole design?

- Confidence intervals
  - Interval which contains true value of parameter with certain probability
  - Point estimate of parameter
    - Does not quantify the quality or range of data collected

$$P(\Theta - \varepsilon_1 < \theta < \Theta - \varepsilon_2) = \gamma$$

- Fault-injection testing can be viewed as series of independent Bernoulli trials
  - Two possible outcomes
    - Success – no observable error
    - Failure – injection causes observable error



Calculation of Confidence Interval

- Chebyshev's inequality $P(\hat{\Theta}-\varepsilon<\theta<\hat{\Theta}-\varepsilon)\geq 1-\frac{Var[\hat{\Theta}]}{\varepsilon^2}$
  - One of simplest ways to obtain a confidence interval
  - Used when variance of estimator (theta) is known or can be easily estimated
  - Prior knowledge of distribution type is not required
  - Quality of bounds can be improved if underlying distribution is known
- Binominal estimation of confidence interval for error rate
  - In case of FI we know underlying distribution
    - Sum of multiple Bernoulli trials is binomially distributed
  - More difficult to calculate as no closed form solution exists
  - Yields bounds which are much tighter than using Chebyshev's inequality

$$B(k_0 : n, p) \leq \frac{1-\gamma}{2}$$

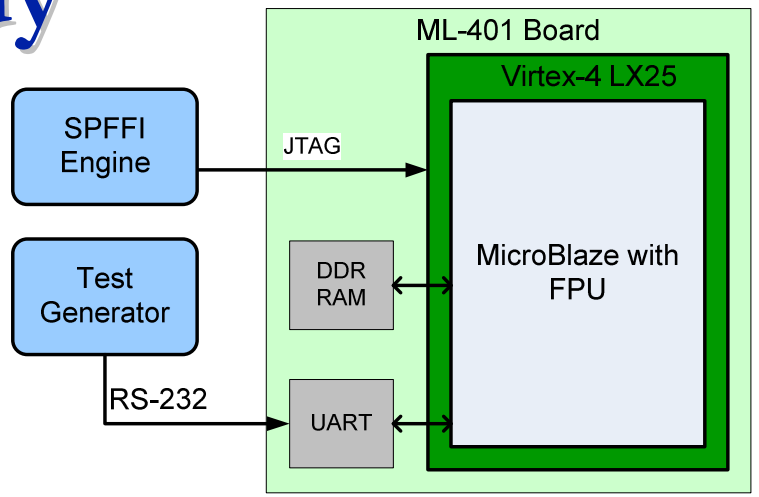
$$1-B(k_1-1 : n, p) \leq \frac{1-\gamma}{2}$$

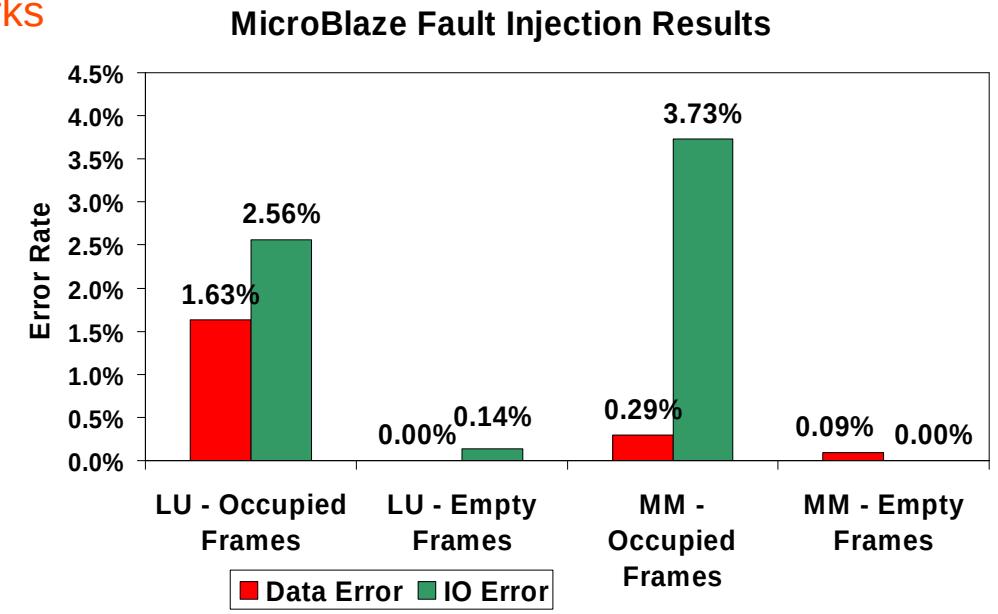
$$k_0 \leq S_n \leq k_1$$

- Using similar approach, it is possible to calculate number of trials required to obtain certain confidence interval

MicroBlaze Case Study

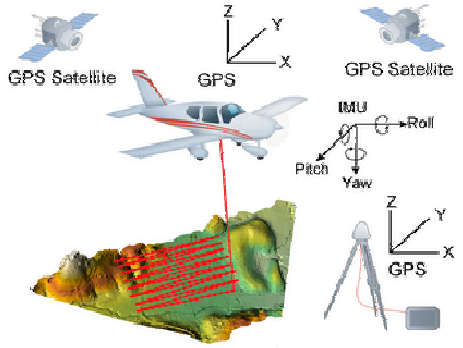
- Soft-core processors are often used as computational resources on FPGA systems
  - What error rate can we expect without FT mitigation?
- Experiment Setup
  - SoC consists of one MicroBlaze with FPU
  - Employs system-level testing methodology
  - 10,000 experiments performed for each campaign
- Test Generator
  - Two popular linear-algebra benchmarks
    - Matrix Multiply
    - LU Decomposition
- Results
  - Occupied Frames
    - LU error rate: 4.19% with 99% CI of [3.67%, 4.72%]
    - MM error rate: 4.02% with 99% CI of [3.51%, 4.54%]
  - Unoccupied Frames
    - LU error rate: 0.14% with 99% CI of [0.04%, 0.25%]
    - MM error rate: 0.09% with 99% CI of [0.01%, 0.19%]

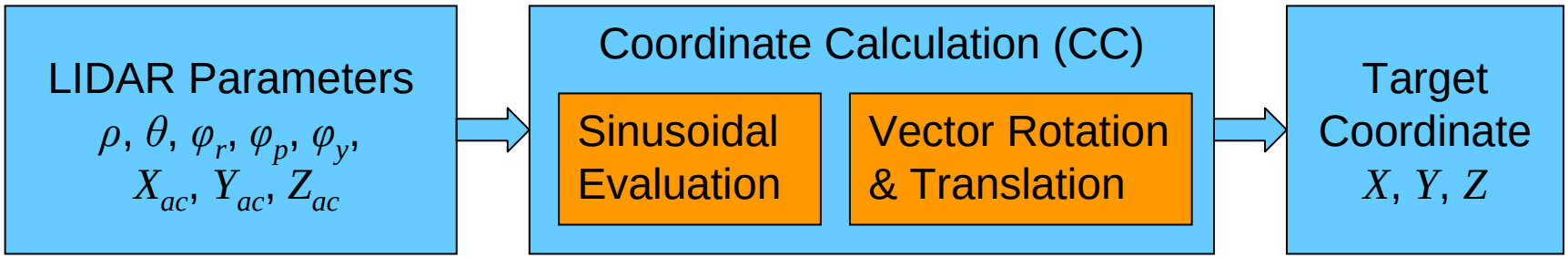




LIDAR Case Study

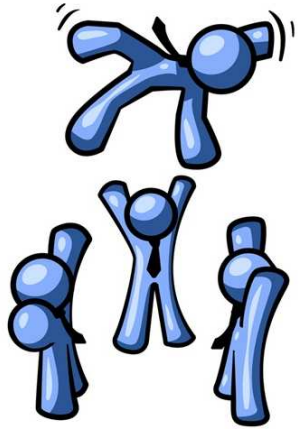
- LIDAR (Light Detection and Ranging)
  - Widely used in remote sensing for terrain mapping
  - Susceptible to upsets due to hazardous operating environment
    - SCP used as error mitigation technique
  - Coordinate Calculation phase constructs 3D information of targets based on set of LIDAR parameters
    - Each laser return is processed independently





- Experimental Setup
  - Similar hardware setup to previous case study
  - Employs module-level testing methodology
  - 20000 test vectors per injection trail
- Results (undetected errors)
  - No FT error rate: 5.18% with 99% CI of [4.61%, 5.77%]
  - SCP error rate: 0.39% with 99% CI of [0.23%,0.57%]

Conclusions and Future Work



- Conclusions
  - SPFFI – new and innovative fault-injection system
    - Simplifies fault testing
    - Does not require specialized hardware
    - Fills niche where other methods fall short
  - Proposed a methodology for fault-injection testing
    - Module- and system-level testing
    - Use of combination of partial and full reconfiguration
  - Complete testing is not always needed
    - Confidence intervals can provide answers which are close enough for development testing
  - Demonstrated multiple case studies exemplifying use of SPFFI
- Future Work
  - Support for Virtex-5 and Virtex-6 devices
  - Augment BRAM injection capabilities
  - Increase fault-injection speed with improved JTAG interface

QUESTIONS?

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant No. EEC-0642422. We also gratefully acknowledge tools provided by Xilinx.

Backup Slides: RFT Case Study

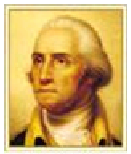
CHREC
NSF Center for High-Performance Reconfigurable Computing

MAPLD 2009

UF UNIVERSITY of FLORIDA

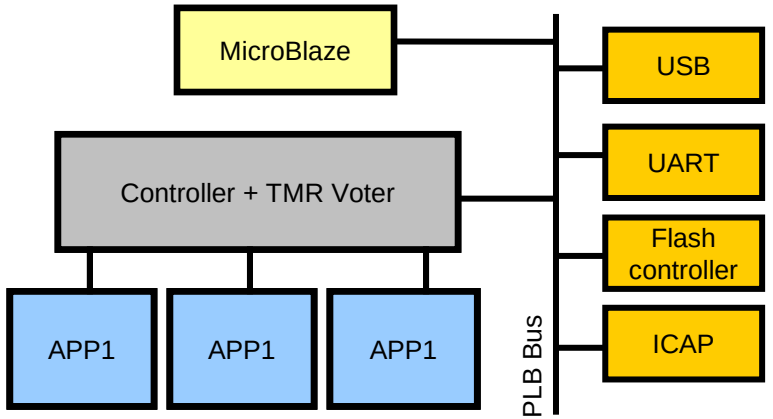
Virginia Tech
VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY

BYU
BRIGHAM YOUNG UNIVERSITY


THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON DC

Reconfigurable Fault Tolerance

- Motivations for RFT
  - Dynamically vary fault-tolerance levels depending on external stimuli
  - Obtain high reliability during critical moments while maximizing performance (or minimizing power) at other times
- Partial Reconfiguration enables system flexibility
  - Ability to move Partial Reconfiguration Modules (PRM) around to different Partial Reconfiguration Regions (PRR)
  - Ability to modify level of fault tolerance in a single PRM
  - Ability to add multiple PRMs to increase fault tolerance through replication
- Possible FT approaches for RFT components
  - Coarse-Grained Replication (SCP, TMR)
  - Algorithm-Based Fault Tolerance (ABFT)
  - Error Correcting Codes (ECC)
  - FT-HLL through source-to-source translation



RFT Case Study

- Two ABFT approaches explored
  - Matrix Multiplication (MM)
  - Linear Transform (LT)
  - Algorithm details in appendix
- Baseline non-FT designs compared to multiple FT approaches
  - **ABFT** - Original VHDL design with additional ABFT components
  - **TMR** – Triplicated version of original
  - **Hybrid** – ABFT design with key components triplicated
- Design reliability tested with SPFFI
  - Single-bit errors injected into configuration memory
    - Focus on LUT and routing resources
  - Errors recorded and categorized for later analysis

- ABFT designs detected >70% of errors from original design
  - Hybrid design had even higher reliability
- ABFT and Hybrid designs approach reliability of traditional TMR
  - However, TMR automatically provides correction
  - ABFT requires additional hardware and cycles to correct erroneous data

	Overhead (Slice / BRAM / Cycles)	Undetected Errors (%)
MM – Original	---	5.75%
MM – ABFT	184% / 0% / 40%	0.76%
MM – Hybrid	196% / 0% / 40%	0.47%
MM – TMR	242% / 200% / 0%	0.66%
LT – Original	---	3.05%
LT – ABFT	149% / 100% / 5%	0.88%
LT – Hybrid	215% / 100% / 5%	0.53%
LT – TMR	284% / 200% / 0%	0.35%

MM – TMR created using VHDL-level replication

LT – TMR created using BYU's EDIF-based TMR tool