Reconfigurable Fault Tolerance (RFT) for FPGA-based Space Computing

CHREC
NSF Center for High-Performance Reconfigurable Computing

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

Grzegorz Cieslewski
Adam Jacobs
Chris Conger
Alan D. George
Brandon Kilpatrick

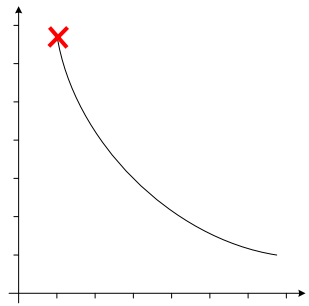
ECE Department, University of Florida

Outline

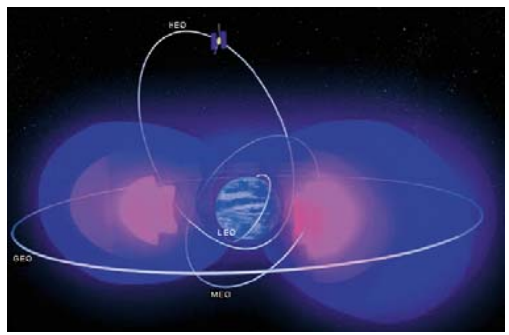
- Introduction
- Taxonomy of FT
- Current FPGA Techniques
- RFT Architecture
- Power Consumption
- Overhead
- Reliability
- Conclusions

Introduction to RFT

- **PROBLEM** – Research how to take advantage of reconfigurable nature of FPGAs, enable dynamically-adaptive fault tolerance (FT) in RC systems
  - Leverage *partial reconfiguration* (PR) where advantageous
  - Explore virtual architectures to enable PR and *reconfigurable fault tolerance* (RFT)



- **MOTIVATIONS** – Why go with fixed/static FT, when performance & reliability can be tuned as needed?
  - Environmentally-aware & adaptive computing is wave of future
  - Achieving power savings and/or performance improvement, *without sacrificing reliability*

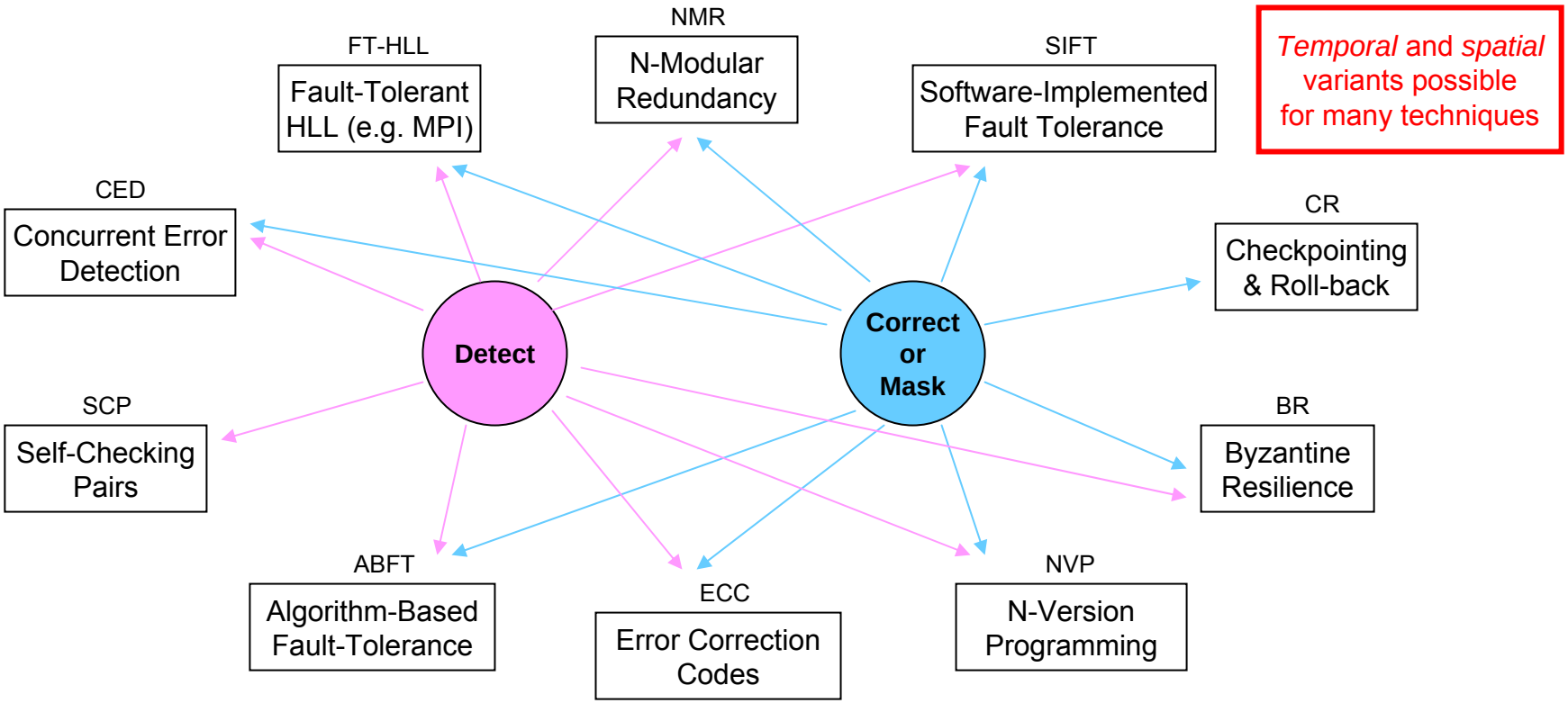


Satellite orbits, passing through the Van Allen radiation belt

- **CHALLENGES** – limitations in concepts and tools, open-ended problem requires innovative solutions
  - Conventional FT methods largely based upon radiation-hardened components and/or fault masking via chip-level TMR
  - Highly-custom nature of FPGA architectures in different systems and apps makes defining a common approach to PR difficult

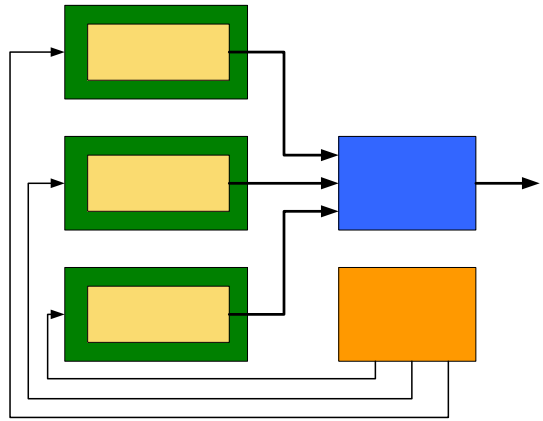
Taxonomy of FT

- First, let us define various possible modes/methods of providing fault tolerance
  - Many options beyond conventional methods of spatial TMR
  - Software FT vs. hardware FT concepts largely similar, differences at implementation level
  - Radiation-hardening not listed, falls under "prevention" as opposed to detection or correction

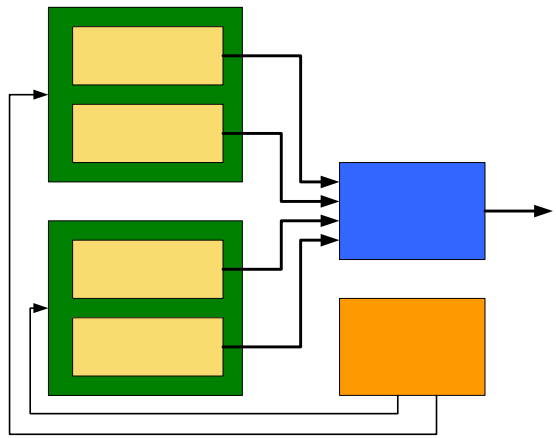


Current FPGA-Based FT Techniques

- Current FT techniques
  - Scrubbing
    - Configuration memory is periodically refreshed to prohibit error accumulation over time
  - External Replication
    - Use of multiple devices – three or more FPGAs connected to external radiation-hardened voter
  - Internal replication of whole design
    - Replicate user module internally on FPGA
      - Can use internal or external voter
      - XTMR
      - BYU EDIF Tools
  - Hybrid Replication
    - Uses both internal and external replication techniques
- Appropriate solution depends upon many factors
  - Expected operating conditions
    - Usually worst-case scenario taken into account
  - Performance requirements
    - Placing multiple user modules on same FPGA can decrease overall performance
  - Power requirements
    - Using multiple FPGAs can significantly increase power consumption of whole design
  - Application characteristics
    - Real-time requirements
    - Uptime requirements



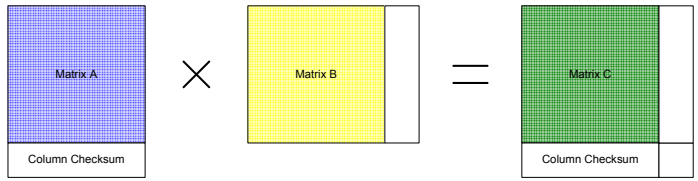
Hardware TMR with scrubbing



Hybrid architecture

Possible FT Modes for RFT Components

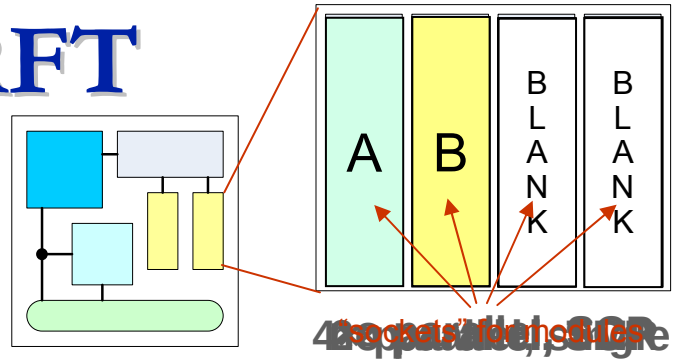
- Coarse-Level Replication
  - Self-Checking Pair (SCP)
    - Two identical components working in tandem
    - Errors can be detected but recovery has to be taken at a higher level (CPU)
  - Triple-Modular Redundancy (TMR)
    - Three identical components processing identical data
    - Recovery can be accomplished by majority voting
- Algorithm-Based Fault Tolerance (ABFT)
  - Suitable for certain linear algebra operations and algorithms that can be expressed in using those operations
  - Augments matrices with extra rows or columns containing weighted checksums
  - Checksums are preserved through the linear operations
- Error-Correcting Codes (ECC)
  - Suitable for buses and memory components
  - Employ extra redundant bits to provide error detection and correction
- FT-HLL through source-to-source translation
  - Designed to provide FT for software running on CPUs
  - Transforms high-level language code into fault-tolerant version by reordering and replicating code fragments
  - Platform- and compiler independent

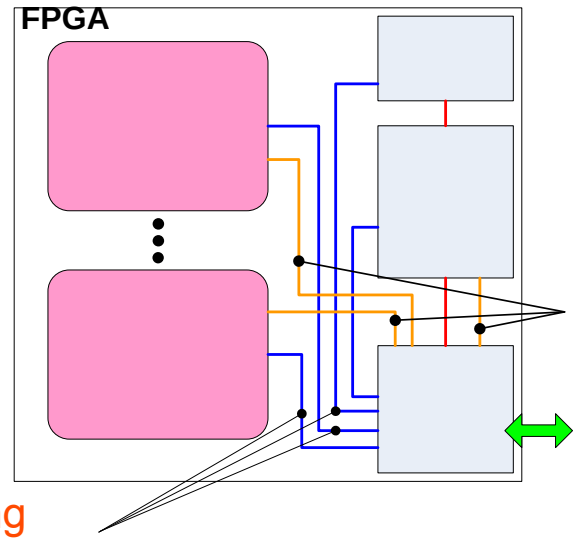



```
1   #pragma S2S start
2   int i = 0;
3   int i__1 = 0;
4   double *data__1 = data;
5   double sum = 0;
6   double sum__1 = 0;
7   for ( i = 0, i__1=0;
8         i < 100 && i__1 < 100;
9         i++ , i__1++)
10  {
11      sum += data[i];
12      sum__1 +=data__1[i__1];
13  }
14  #pragma S2S stop
15  if(i!=i__1)
16      error();
17  if(sum!=sum__1)
18      error();
```

Virtual Architecture for RFT

- Novel concept of *adaptable component-level protection (ACP)*
- Common components within VA:
  - Multiple Reconfigurable Regions
    - Largely module/design-independent
  - Error Status Register (ESR) for system-level error tracking/handling
  - Synchronization controller, for state saving and restoration
  - Configuration controller, two options:
    - Internal configuration through ICAP
    - External configuration controller
- Benefits of internal protection:
  - Early error detection and handling = faster recovery
  - Redundancy can be changed into parallelism
  - Redundancy/parallelism can be traded for power
  - PR can be leveraged to provide uninterrupted operation of non-failed components
- Challenges of internal protection:
  - Difficult to eliminate single points of failure, may still need higher-level (external) detection and handling
  - Stronger possibility of fault/error going unnoticed
  - Single-event functional interrupts (SEFI) are concern

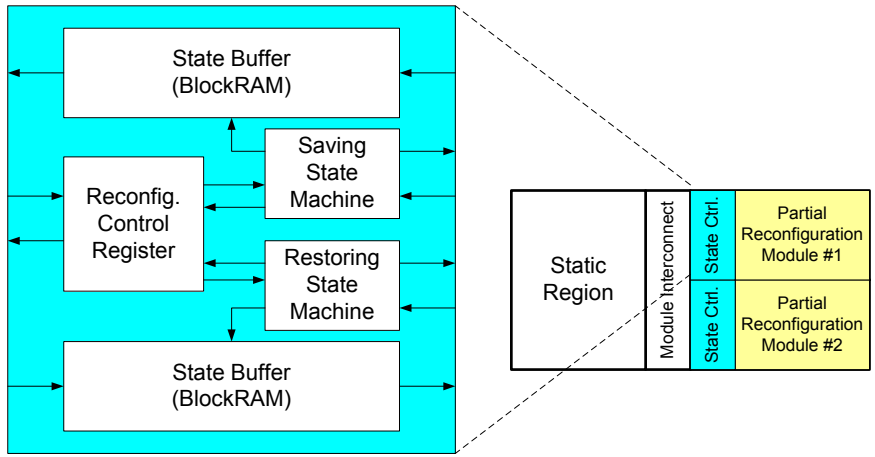




VA concept diagram

RFT Architecture

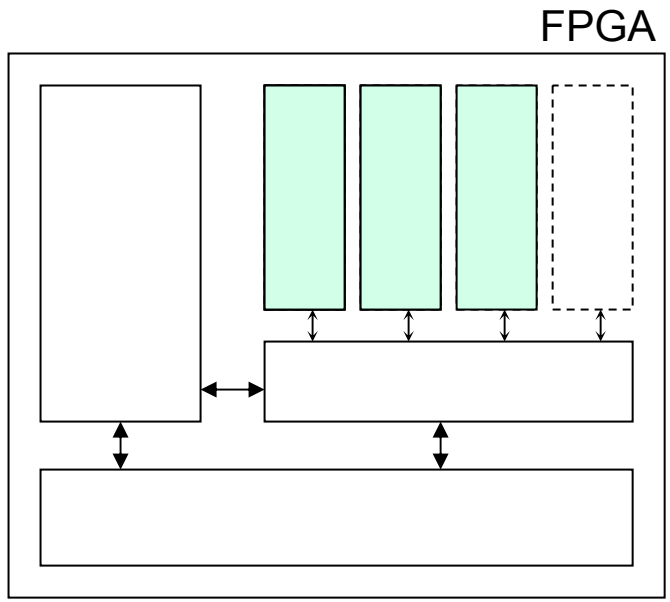
- Partial Reconfiguration (PR) enables system flexibility
  - Ability to move Partial Reconfiguration Modules (PRM) around to different Partial Reconfiguration Regions (PRR)
  - Ability to modify level of fault-tolerance in a PRM
  - Ability to add multiple PRMs to increase fault tolerance through replication



- Two Possible Approaches
  - Create multiple PRMs for a given function representing different levels of fault tolerance
    - Swap entire module when changing protection levels
    - No protection, SCP, TMR
  - Create a single PRM and use multiple copies to add fault tolerance
    - An additional voter module is used to compare outputs between modules
- Explicit State Saving
  - Module designer adds functionality to record and update all state variables
    - Reconfiguration Control Register (RCR) instructs modules to save any data needed to restore state
    - RCR also interfaces with system's Configuration Controller
    - Allows continuous operation while changing a PRM fault-tolerance level
- Configuration controller can store multiple module states off-chip
  - Controller is a main component of a traditional Partial Reconfiguration framework

Bitstream Relocation

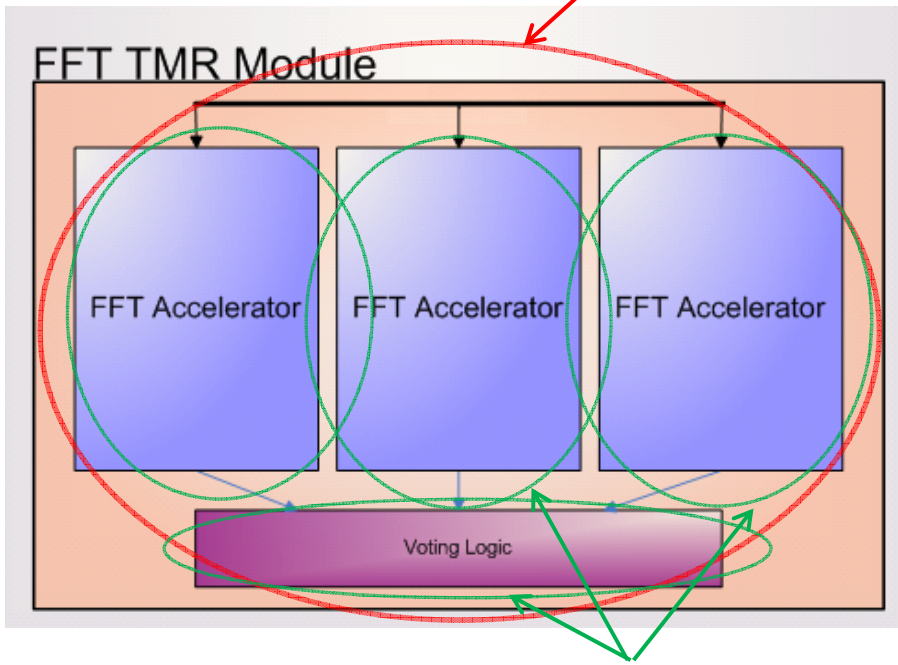
- Bitstream relocation
  - Changing frame addresses and bitstream composition to move (or replicate) physical location of a module on chip
  - Relocation can only be performed with partial bitstreams
  - Advantages
    - Increases flexibility in time-multiplexing FPGA resources
    - Reduce bitstream storage requirements
    - Migration of bitstream to other FPGAs
    - Ability to move modules away from faults
- Results
  - Bitstream parser written in C
  - Currently executed off-line on workstation
  - Next being ported to embedded PPC/Microblaze or host processor

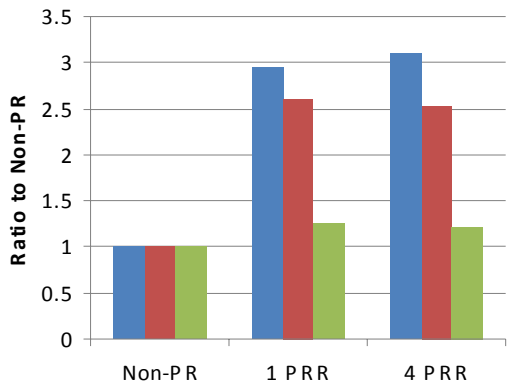


Overhead of PR

- Illustrate effect of breaking same design up into different number of PRRs
- Generally speaking, required resources increase when going from non-PR to PR
  - Slices increase ~200% with PR
  - BRAMs increase ~150% with PR
  - DSPs increase ~25% with PR
- Take-away points
  - Largest price paid by making PR, period
  - Decomposing PR design into multiple PRRs comes at much less significant cost than non-PR vs. PR
  - From FT perspective, physical isolation decreases chances of single fault affecting multiple modules
  - From general PR perspective, more/smaller regions equate to lower reconfiguration overhead

	Non-PR	1 PRR	4 PRR
Slice Registers	11556	43120	45344
Slice LUTs	10196	86240	90688
Slices	3657	10780	11310
BRAMs	23	60	58
DSPs	48	60	58





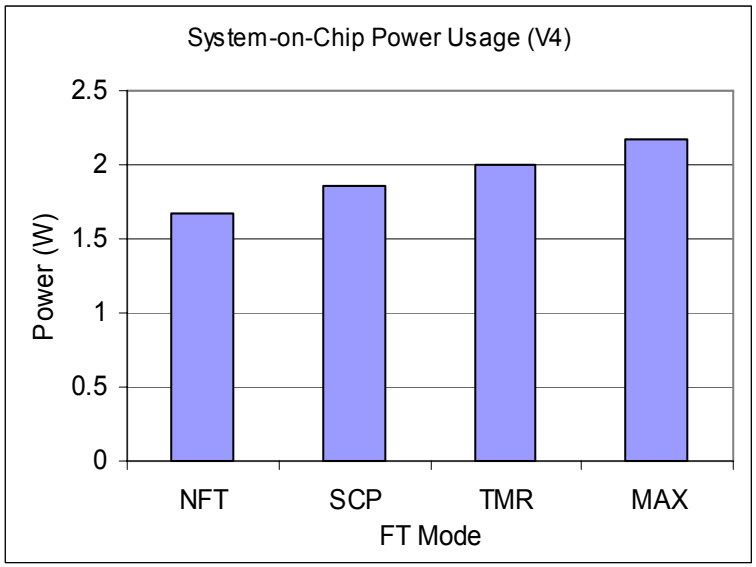
Situation will vary by app… these results believed to be close to worst-case

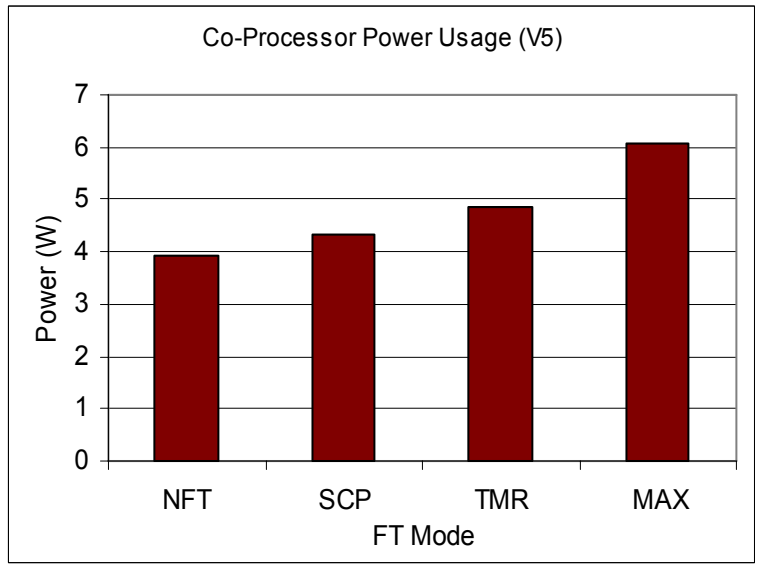
Power / Overhead Analysis

Using spatial TMR & SCP, assuming 25% activity rate

- Resource Utilization
  - SoC – ~2.3× resource requirement for MAX over None
  - Co-processor – ~3.8× resource requirement for MAX
- Power consumption
  - SoC – higher FT increases power 10-30%
  - Co-processor – higher FT increases power 10-50%
- Max case uses all four slots of RFT VA
  - e.g. two parallel instances of SCP, 4-way parallel operation
  - "Mode" not relevant to power consumption, simply depends upon how many slots are populated & active

	System-on-Chip (V4FX20)				Co-Processor (V5SX95)			
	None	SCP	TMR	MAX	None	SCP	TMR	MAX
Registers	3750	5325	6886	8444	11317	21904	32290	43077
LUTs	3528	5059	6564	8017	11033	21563	32285	42642
BRAMs	7	10	13	16	39	78	117	156
DSPs	3	6	9	12	44	88	132	176





Analytical Reliability Analysis

- Analytical reliability analysis can help estimate fault susceptibility of proposed designs
  - Most important parameters are “upset rates”, or lambdas (λ) for each component of RFT; can be approximated based upon respective components resource utilization
  - Overall system reliability can be expressed as a product of component reliabilities
  - Component-level reliability expression may change depending upon current mode of fault tolerance
  - Currently, static part of design is not protected by any FT technique
- MTTF is a one of important reliability metrics
  - Preliminary results show that possible to significantly increase MTTF using component-level protection in RFT
  - SCP is more susceptible to upsets and functional interrupts but allows for better error detection than case without FT

Example Expressions

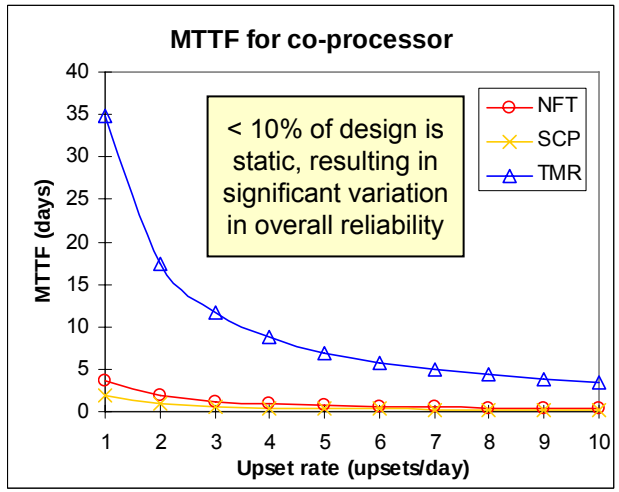
$$R_{BASE}(t) = e^{-\lambda_{\mathrm{mod}} \cdot t}$$

$$R_{SCP}(t) = e^{-\lambda_{vote} \cdot t} \cdot e^{-2\lambda_{\mathrm{mod}} \cdot t}$$

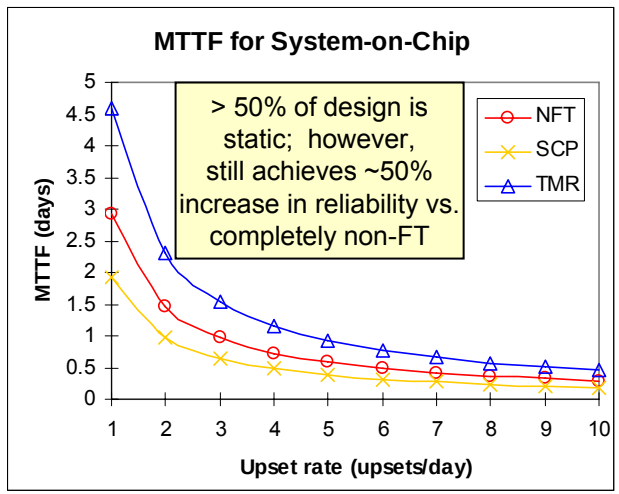
$$R_{TMR}(t) = e^{-\lambda_{vote} \cdot t} \cdot (3e^{-2\lambda_{\mathrm{mod}} \cdot t} - 2e^{-3\lambda_{\mathrm{mod}} \cdot t})$$

$$R_{ECC}(t) = e^{-\lambda_{codec} \cdot t} \cdot \left[e^{-n \cdot \lambda_{bit} \cdot t} + e^{-(n-1) \cdot \lambda_{bit} \cdot t} - n \cdot e^{-n \cdot \lambda_{bit} \cdot t}\right]^m$$

$$R_{overall}(t) = \prod_i R_i(t) \qquad MTTF = \int_0^\infty R_{overall}(t)\,d(t)$$



MTTF for co-processor architecture



MTTF for SoC architecture

Conclusions and Future Work

- Fault-tolerant computing for space should be more versatile and adaptive than merely RadHard & spatial TMR
  - Fixed, worst-case designs are extremely limiting
    - Higher power consumption
    - Large area overhead
  - Instead, variety of techniques from FT taxonomy can be employed
    - SCP, ABFT, ECC, etc. can reduce required overhead while maintaining reliability
  - Adaptive systems (via RFT) can react to environmental changes
- Future Work
  - Extend and refine concept of RFT
  - Develop proposed RFT architectures
  - Extend analytical reliability analysis of proposed RFT architectures
  - Verify and augment analytical reliability analysis using fault injection

Acknowledgements

This research was made possible by

- NSF I/UCRC Program (Center Grant EEC-0642422)
- CHREC members (31 industry & govt. partners)
- Honeywell (prime contractor for NASA's DM)
- Xilinx (donated tools)

Questions?

Please visit CHREC Booth for general info on CHREC mission, projects, schools, and members