Information Sciences Institute

Radiation Hardening of FPGA-embedded CPUs via Software, Validated with Fault Emulation

John Paul Walters, Kenneth M. Zick, Matthew French

USC/ISI

USC

USC Viterbi School of Engineering

FPGAs Today

FPGAs have evolved, becoming heterogeneous

— hard CPU cores, Ethernet cores, Giga-bit transceivers

Legacy features (known mitigation techniques)

New features

Virtex-5 FXT FPGA Platform
Optimized for Embedded Processing with High-Speed Serial Connectivity (1.0 Volt)

	Port Number	XC5VFX30T	XC5VFX70T	XC5VFX100T	XC5VFX130T	XC5VFX200T
	EasyPath™ Cost Reduction Solutions [1]	—	XCE5VFXT70T	XCE5VFXT100T	XCE5VFXT130T	XCE5VFXT200T
Logic Resources	Slices [2]	5,120	11,200	16,000	20,480	30,720
	Logic Cells [3]	32,768	71,680	102,400	131,072	196,608
	CLB Flip-Flops	20,480	44,800	64,000	81,920	122,880
Memory Resources	Maximum Distributed RAM (Kbits)	380	820	1,240	1,580	2,280
	Block RAM/FIFO w/ECC (36Kbits each)	68	148	228	298	456
	Total Block RAM (Kbits)	2,448	5,328	8,208	10,728	16,416
Clock Resources	Digital Clock Managers (DCM)	4	12	12	12	12
	Phase Locked Loop (PLL)/PMCD	2	6	6	6	6
I/O Resources [4]	Maximum Single-Ended Pins	360	640	680	840	960
	Maximum Differential I/O Pairs	180	320	340	420	480
	I/O Standards	HT, LVDS, LVDSEXT, RSDS, BLVDS, ULVDS, LVPECL, LVCMOS33, LVCMOS25, LVCMOS18, LVCMOS15, LVTTL, PCI33, PCI66, PCI-X, GTL, GTL+, HSTL I (1.2V,1.5V,1.8V), HSTL II (1.5V,1.8V), HSTL III (1.5V,1.8V), HSTL IV (1.5V,1.8V), SSTL2 I, SSTL2 II, SSTL18 I, SSTL18 II				
Embedded [5] Hard IP Resources	DSP48E Slices	64	128	256	320	384
	PowerPC® 440 Processor Blocks	1	1	2	2	2
	PCI Express Endpoint Blocks	1	3	3	3	4
	10/100/1000 Ethernet MAC Blocks	4	4	4	6	8
	RocketIO™ GTP Low-Power Transceivers	—	—	—	—	—
	RocketIO™ GTX High-Speed Transceivers	8	16	16	20	24

Xilinx V5FXT Datasheet

FPGA Embedded PowerPC core outperforms radiation hardened RISC processors

Processor	Mongoose V	RAD6000	RAD750	Virtex4 PPC405	Virtex 5 PPC440
Dhrystone MIPS	8	35	260	900	2,200

Can RHBSW techniques be developed for hard CPU cores?

Existing Embedded PPC Fault Tolerance Approaches

Problem: PowerPC state is dynamic & cannot be protected by configuration bit scrubbing

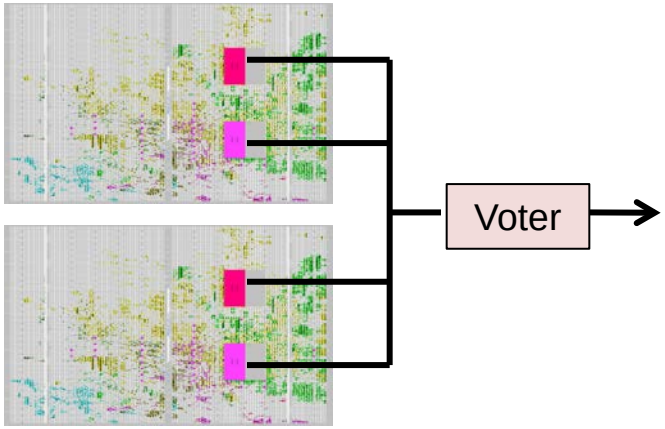
- Fault injection not feasible by this method either

Quadruple Modular Redundancy

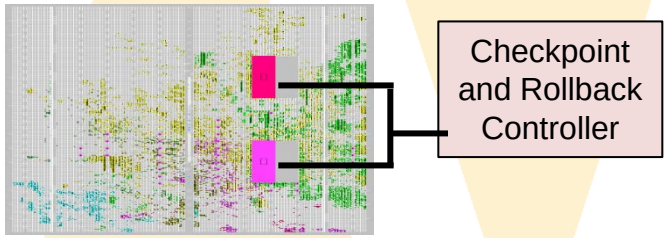
- 2 Devices = 4 PowerPC cores
- Vote on result every clock cycle
- Fault detection and correction
- ~300% Overhead

Dual Processor Lock Step

- Single device solution
- Error detection only
- Checkpointing and Rollback to return to last known safe state
- 100% Overhead
- Downtime while both processors rolling back



QMR Approach



Dual Lock Step Approach

New fault tolerance techniques and validation methods must be researched.

High Performance Computing Insights

HPC community has similar problem

- 100's to 1000's of nodes
- Long application run times (days to weeks)
- A node will fail over run time

HPC community does not use TMR

- Too many resources for already large, expensive systems
- Power = $

HPC relies more on periodic checkpointing and rollback

Can we adapt these techniques for embedded computing?

- Checkpoint frequency
- Checkpoint data size
- Available memory
- Real-time requirements

Cray HPC System

Fault Tolerance Strategies

Current system consists of:

- Checkpoints, control flow assertions, and heartbeats.

Checkpoint/rollback

- Used to react to a detected fault.

Control flow assertions

- Detect errors in control flow at the application-level.

Heartbeat monitoring

- Detect “liveness” of a PPC.

Checkpoint and Rollback

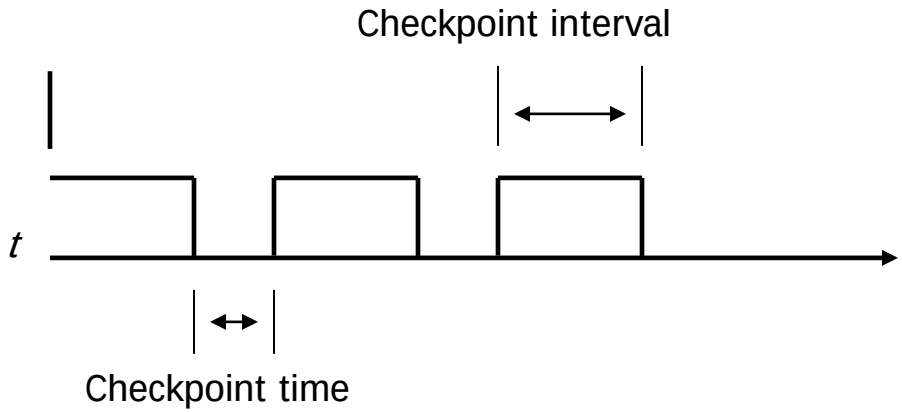
User-level checkpoint / rollback

General purpose

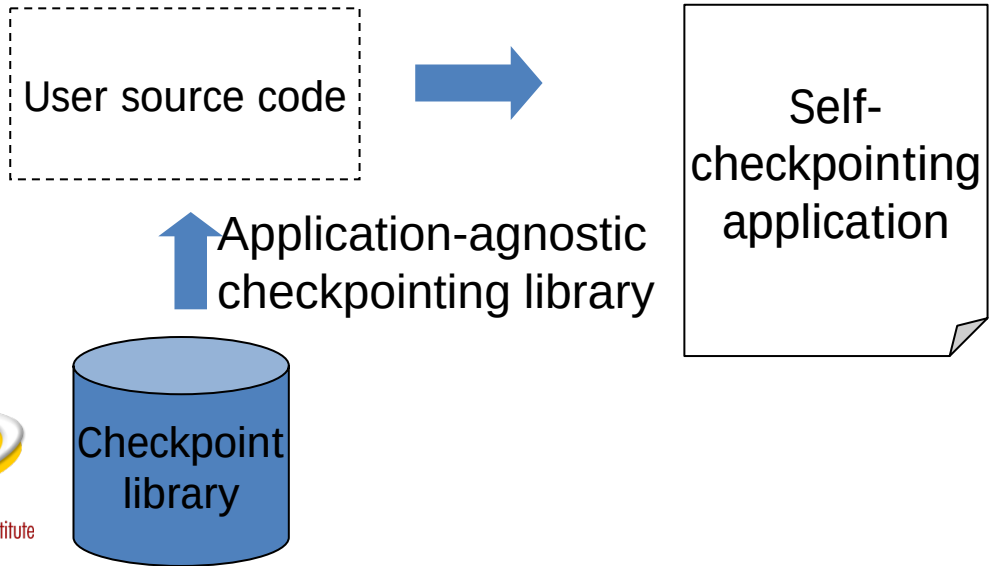
Provides user-defined callbacks

- Helpful for graceful cleanup of files, networks, FPGA fabric

Enables rapid context switching



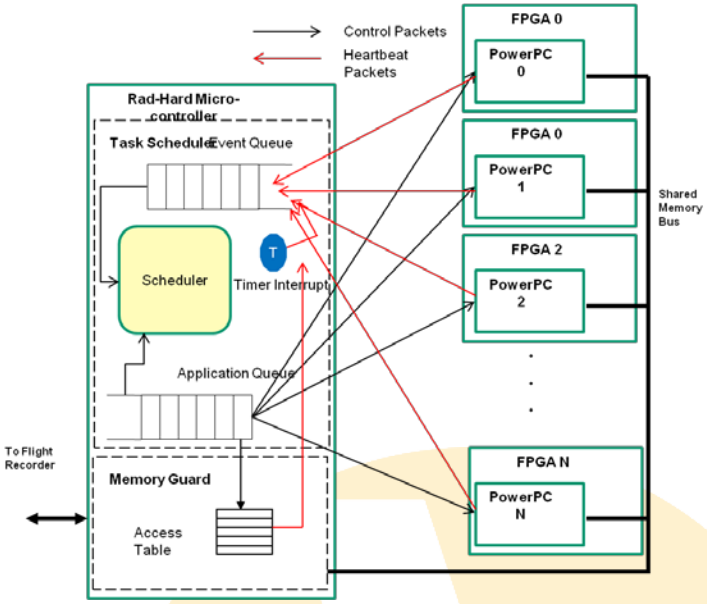
Balance checkpoint interval to upset rate



- User links in checkpoint library
- Library provides checkpoint() and restart() functions
- User inserts calls to checkpoint() at desired location(s)
- Checkpoint to DRAM, BRAM, or compact flash

Heartbeats

- Heartbeats are generated by an FPGA based timer interrupt
- Each Heartbeat includes at least the following:
  - Destination ID / Source ID (1 byte)
  - Message Number (1 byte)
  - Message Type (1 byte)
- Heartbeats output when:
  - Program Starts
  - Program Ends
  - Autonomous Events
  - User-defined interval




```
// On a Timer Interrupt
msg[0] = (PPC_ID<<4) |
          RAD_HARD_ID;
msg[1] = heartbeat_number++;
msg[2] = HEARTBEAT_TYPE;
msg[3] = DATA_LENGTH_ZERO;
Send_Message(msg);
```

Control Flow Assertions

- **Tag blocks of code with signatures**
- **As code progresses check signatures against expected value**
- **Programmer indicates where to put assertions**

Original Code

```
x = 50;
if (condition == 1)
    new_x = x-5;
else
    new_x = x – 3;
z = new_x – x;
```

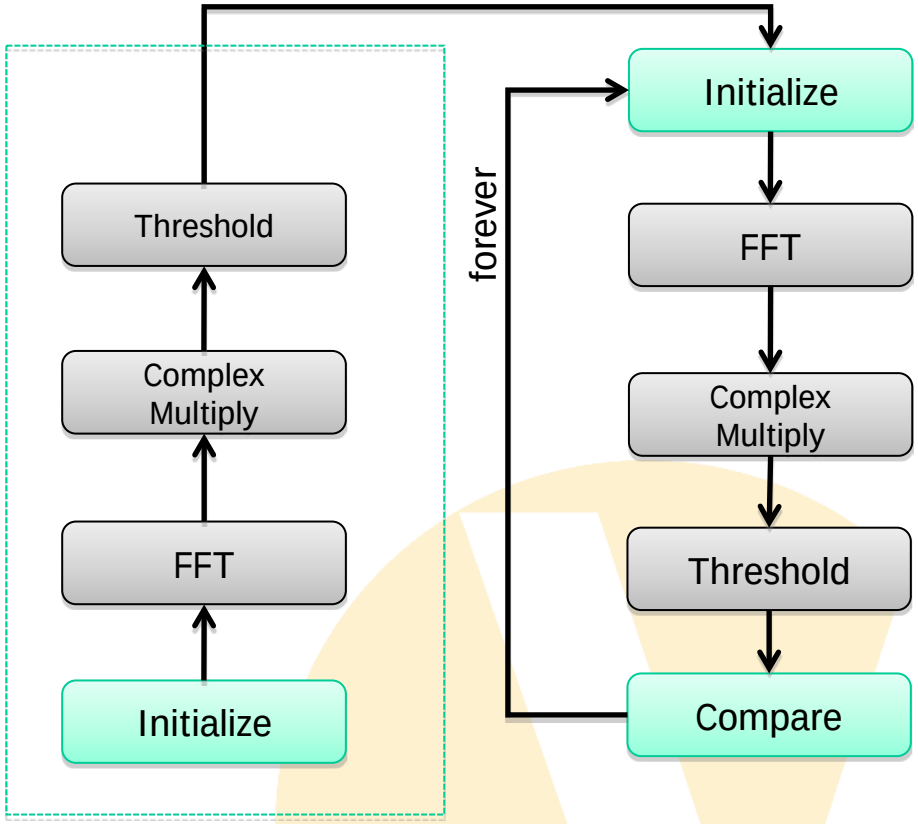
Transformed Code

```
ES_1 = ES_1 ^ 01;
x = 50;
if (condition == 1)
{
    ES_1 = ES_1 ^ 010;
    new_x = x-5;
}else{
    ES_1 = ES_1 ^ 010;
    New_x = x – 3;
}
ES_1 = ES_1 ^ 0100;
if (ES_1 != 0111) error();
z = new_x – x;
```

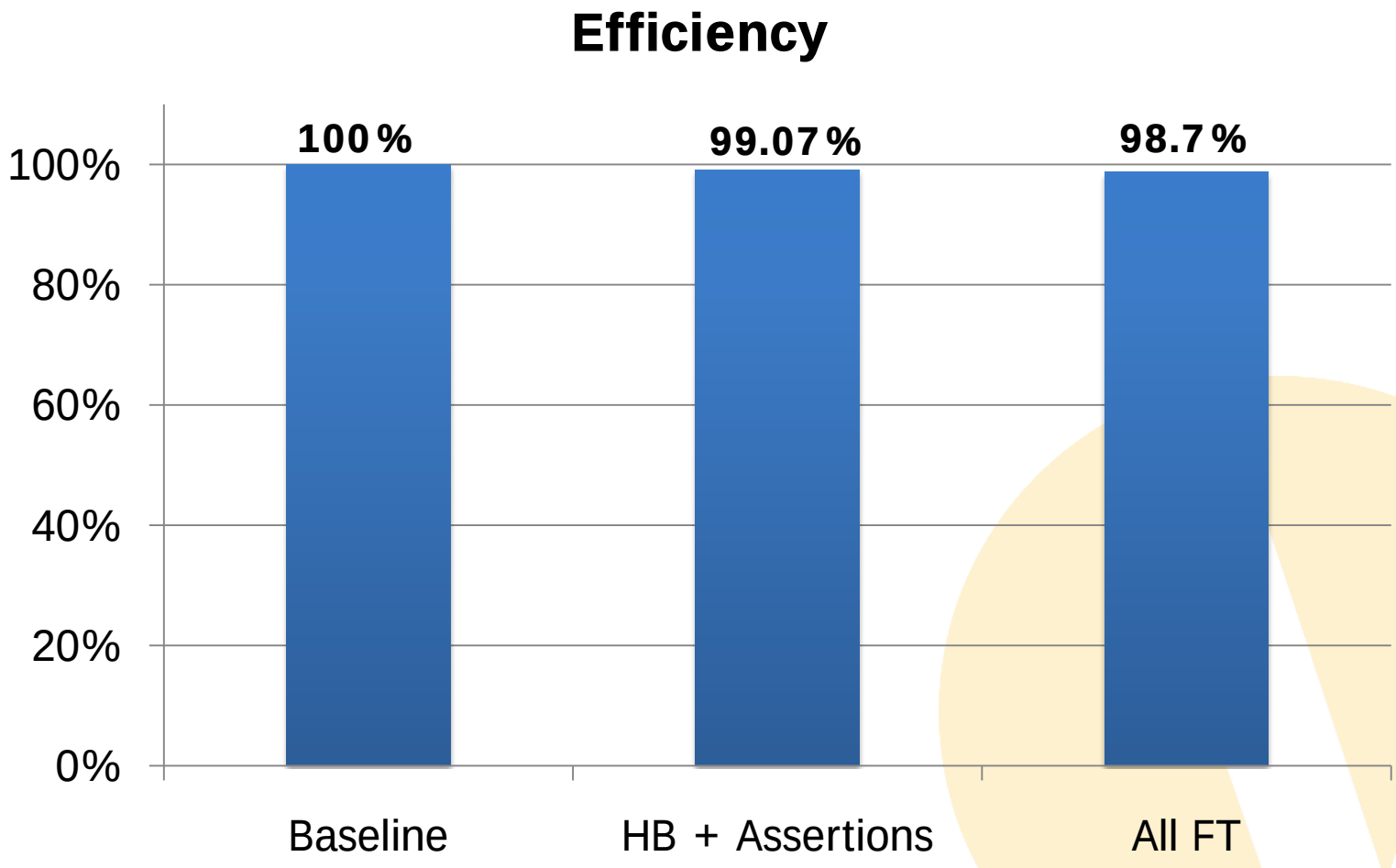
- When an error is detected, alert heartbeat and initiate a rollback.
- Coordinate rollback/restart with 2ndPPC if necessary.

Test Application

- **Combine the major elements of SAR and hyperspectral and loop infinitely over the computation.**
- **From SAR we use FFT and complex multiply.**
- **Use thresholding over the resulting FFT/complex multiply.**
  - Representative of a common hyperspectral classifier.
- **Compare the results to the "golden" output after each computation.**



Efficiency of Fault Tolerance Implementation of Test Application



$$\text{Efficiency} = \left(\frac{Time_{NoFT}}{Time_{FT}}\right) * 100$$

Fault Injection and Future In-Situ Evaluation

Extensive verification through fault emulation, laser testing, and flight test.

On-orbit flight test on MISSE7 experiment

- PowerPC 405 core on commercial Xilinx Virtex-4FX device
- Using the Space Cube 1.0 platform.
- Currently uplinking to ISS
- Allocated a single PPC for test and a shared control PPC.

Fault emulation

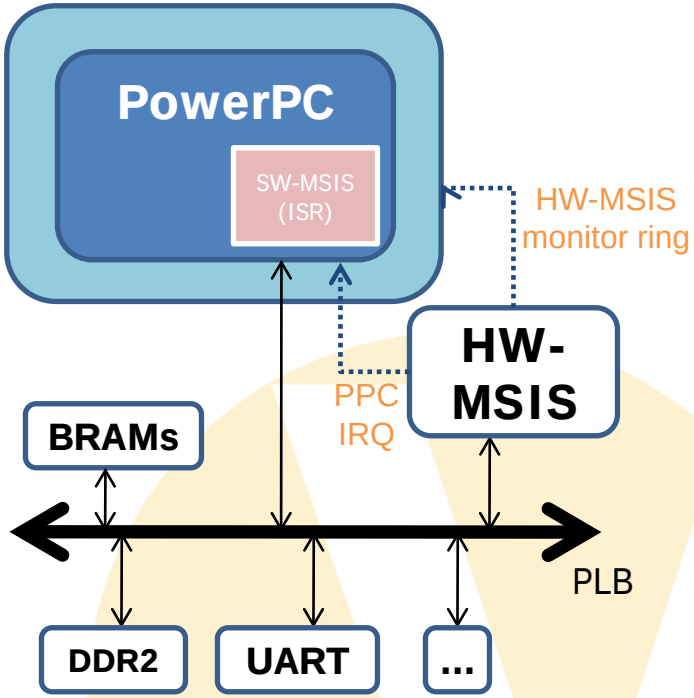
- MSIS: Memory Sentinel and Injection System.
- In use currently to test registers and caches.
- Design closely mirrors the allocated MISSE7 V4-based design.

Laser test in September

- Will correlate fault emulation results with laser test results

MSIS Design

- **MSIS software ISR (SW-MSIS) modifies the registers and performs setup of other corruptions**
  - Effective for SPR and GPR
- **MSIS configurable logic (HW-MSIS) is responsible for generating periodic interrupts and monitoring / modifying PowerPC transactions**
  - Modify cache contents
  - Protect memory regions



Current Test Data

Have compared our fault tolerant design to a base implementation with limited fault tolerance.

Both full injection (registers and cache) and registers-only injection performed

Analyzed 15,000 injections for each injection campaign

Interpreting Fault Injection Data

We broadly classify injection results into 3 categories:

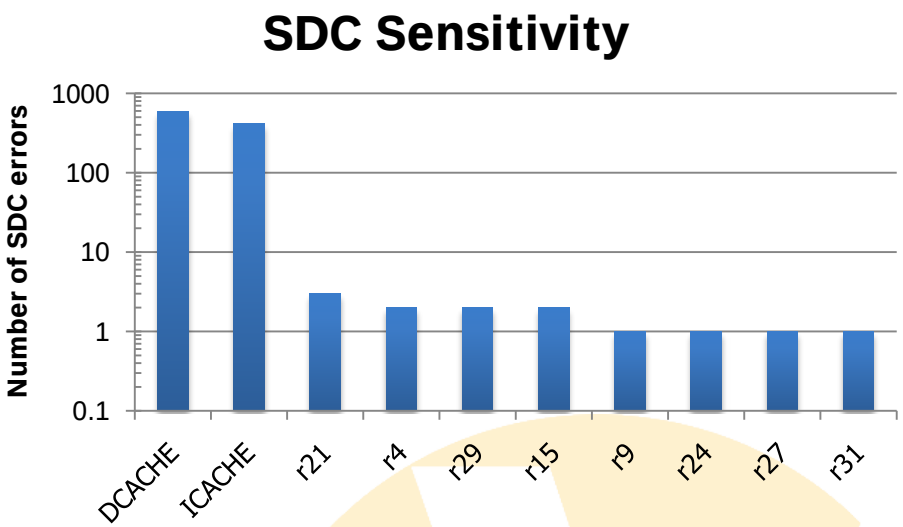
- Good data: an injection occurred and no failure was observed.
- Silent data corruption: an injection occurred which resulted in an incorrect value at the data comparison stage.
  - *Not currently distinguishing between persistent and transient errors*
- Hang: an injection occurred which would typically hang the PPC.

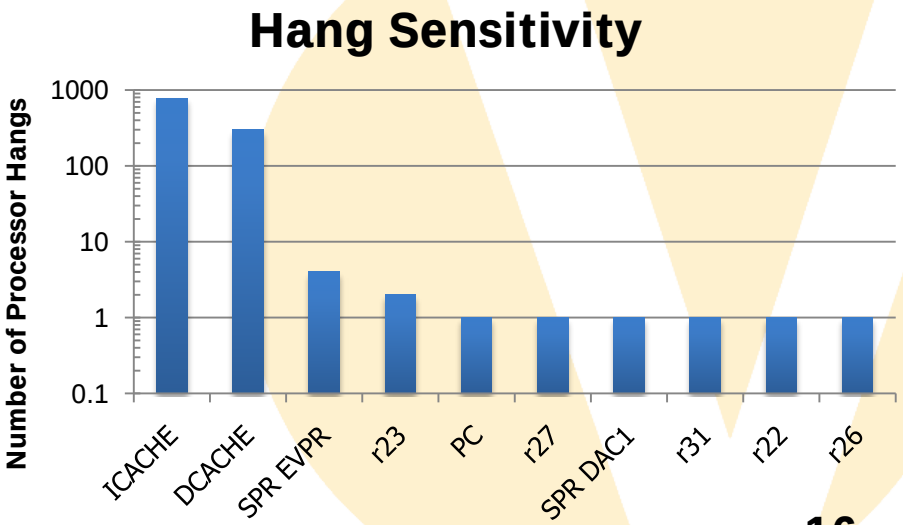
In our FT design we add two subcategories:

- Recovery via rollback: an injection occurred that would have resulted in a PPC hang or silent data corruption, but we were able to roll back and recover from the error.
- Self-reset: an injection occurred that resulted in a processor hang, but the DUT's watchdog timer detected the hang and reset the processor.

Full Injection Results

Result	No FT (%)	FT (%)
Good Data	86%	86.7%
SDC	6.8%	6.8%
Hang	7.2%	0%
Self-reset	0%	1.6%
Recovery via rollback	0%	4.9%
Total Good	**86%**	**91.6%**



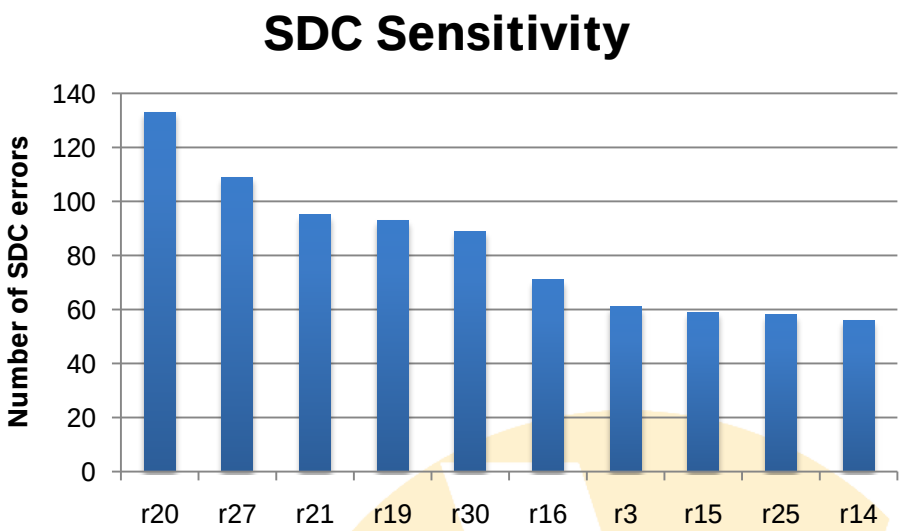


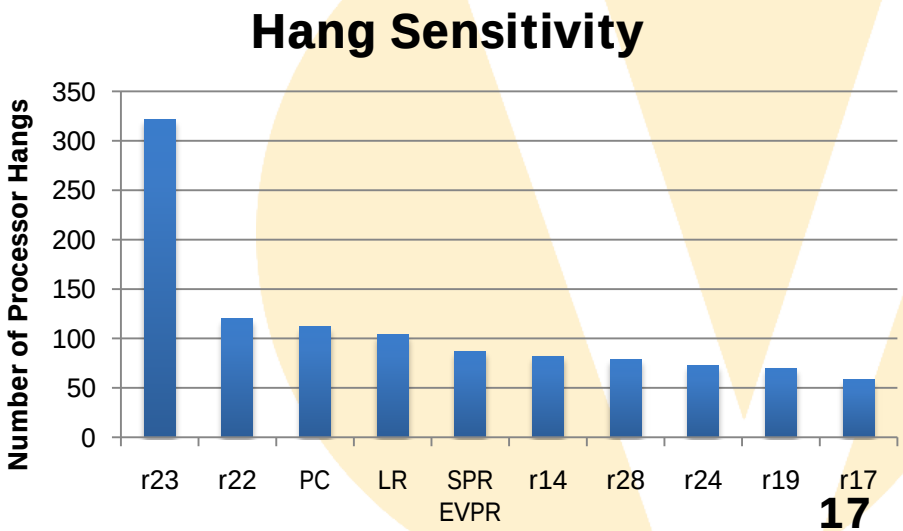
Assuming methods of detecting data corruption, we could reach up to 97.4% fault coverage using a cascading rollback.

Register-only Injection Results

Result	No FT (%)	FT (%)
Good Data	82.4%	78.8%
SDC	8.3%	8.9%
Hang	9.3%	0%
Self-reset	0%	5.0%
Recovery via rollback	0%	7.3%
Total Good	**82.4%**	**86.1%**

Assuming methods of detecting data corruption, we could reach up to 95% fault coverage using a cascading rollback.





Fault Injection Summary Results

Our base techniques result in nearly 92% fault coverage with less than 2% execution time overhead.

- Area overhead: 64KB of BRAM for checkpointing, plus timer and interrupt controller in fabric for heartbeats
- Compared to TMR: 100% error coverage for ~200% area overhead.

With addition of data corruption detection we could achieve up to 97.4% fault coverage

- Current work targets low overhead application-specific error detection.

Next steps: laser testing

Validation of fault tolerance methods using pulsed-laser at Naval Research Lab

- Code runs on PowerPC 405 core in a commercial Virtex-4FX
- In addition to control flow assertions and heartbeat watchdog, we've developed low-cost self-checks for data corruption

Laser testing allows more sophisticated fault model:

- Includes faults in non-SW visible state: CPU pipeline stages, instruction buffer, etc.
- Allows faults during application execution as opposed to pausing the application with an interrupt
- Includes single event transients as well as SEUs

Setup includes custom board, FPGA-FPGA interface, dynamic selection of single laser pulses using a shutter

SUMMARY

Developing a library of fault tolerance routines available to NASA community

- Targeted for science data processing

Initial tests promising

- Our fault tolerance approach introduces minimal overhead

Upgrading Fault Injection

- Developing new techniques to inject faults into the FPGA fabric within the MSIS system.

Test Plans

- Laser testing September 2011
- ISS testing on MISSE-7