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

John Paul Walters, Kenneth M. Zick, Matthew French
USC/ISI
FPGAs have evolved, becoming heterogeneous
— hard CPU cores, Ethernet cores, Giga-bit transceivers

Legacy features (known mitigation techniques)

New features

FPGA Embedded PowerPC core outperforms radiation hardened RISC processors

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

New fault tolerance techniques and validation methods must be researched.
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
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.
User-level checkpoint/rollback

General purpose
Provides user-defined callbacks
- Helpful for graceful cleanup of files, networks, FPGA fabric

Enables rapid context switching

Checkpoint and Rollback

Checkpoint interval

Balance checkpoint interval to upset rate

User source code

Application-agnostic checkpointing library

Self-checkpointing application

- 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 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

```c
// 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);
```
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.
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

Efficiency = \left( \frac{Time_{NoFT}}{Time_{FT}} \right) \times 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
MSI S software ISR (SW-MSI S) modifies the registers and performs setup of other corruptions
  - Effective for SPR and GPR

MSI S configurable logic (HW-MSI S) is responsible for generating periodic interrupts and monitoring/ modifying PowerPC transactions
  - Modify cache contents
  - Protect memory regions
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.
Assuming methods of detecting data corruption, we could reach up to 97.4% fault coverage using a cascading rollback.
### Register-only Injection Results

<table>
<thead>
<tr>
<th>Result</th>
<th>No FT (%)</th>
<th>FT (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good Data</td>
<td>82.4%</td>
<td>78.8%</td>
</tr>
<tr>
<td>SDC</td>
<td>8.3%</td>
<td>8.9%</td>
</tr>
<tr>
<td>Hang</td>
<td>9.3%</td>
<td>0%</td>
</tr>
<tr>
<td>Self-reset</td>
<td>0%</td>
<td>5.0%</td>
</tr>
<tr>
<td>Recovery via rollback</td>
<td>0%</td>
<td>7.3%</td>
</tr>
<tr>
<td><strong>Total Good</strong></td>
<td><strong>82.4%</strong></td>
<td><strong>86.1%</strong></td>
</tr>
</tbody>
</table>

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.
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