XILINX®

Mitigation, Design Flow and Troubleshooting a Soft Processor in a Complex FPGA

Greg Miller[1], Carl Carmichael[1], Gary Swift[1], Chen Wei Tseng[1]

MAPLD 2008

[1]Xilinx, Inc., San Jose, CA/ Longmont, CO

Objectives

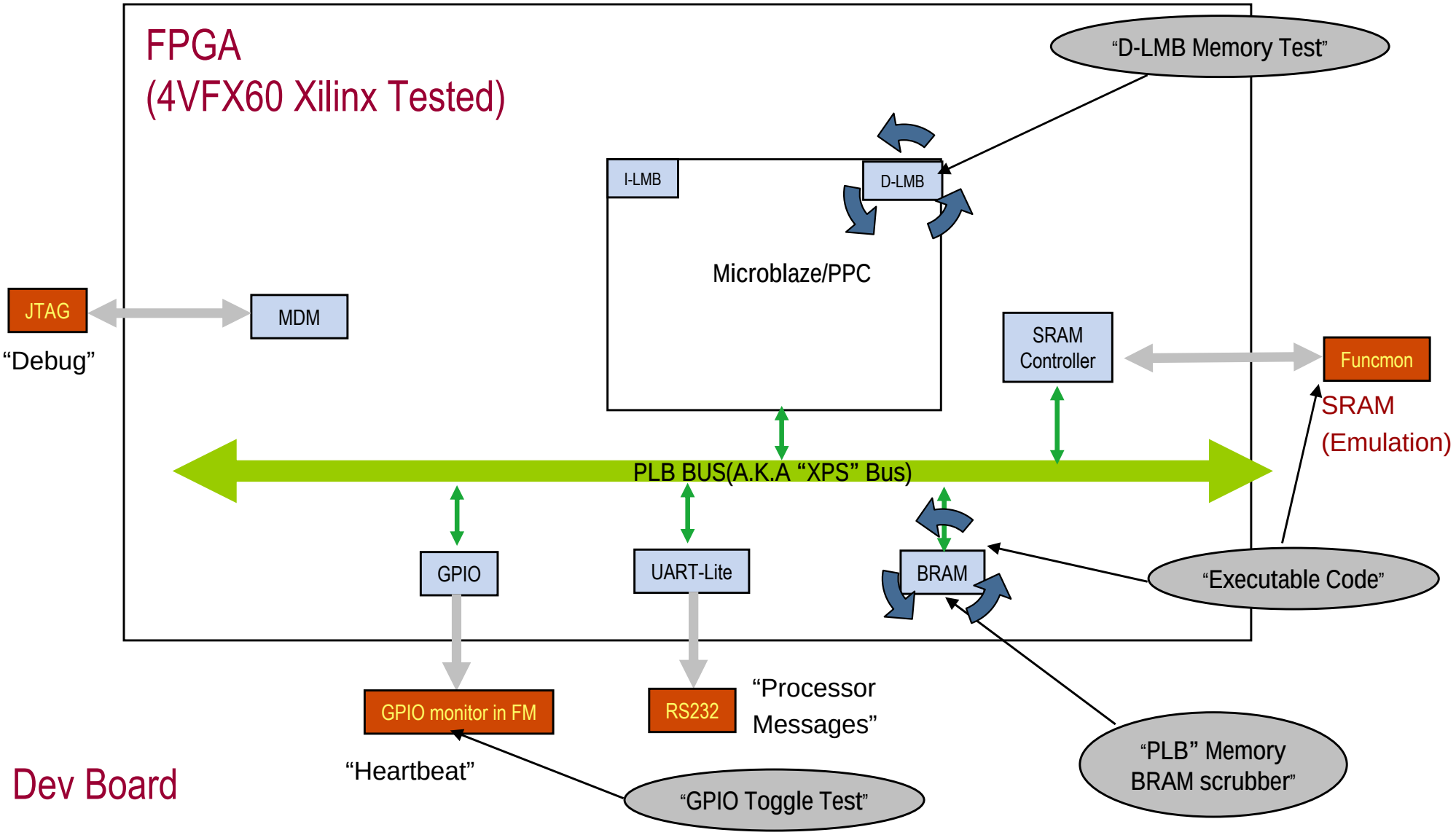
Create a robust soft processor design in a complex FPGA using existing tools.

- Mitigated Design Flow
  - Create a repeatable design flow (full mitigation) using a Soft Processor based system

- Simulation of Sensitivity in Radiation Environment using dynamic tests
  - Use fault injection (on hardware) to simulate and troubleshoot potential problems
  - Dynamic test design using a Simple IO test (Processor Driven)

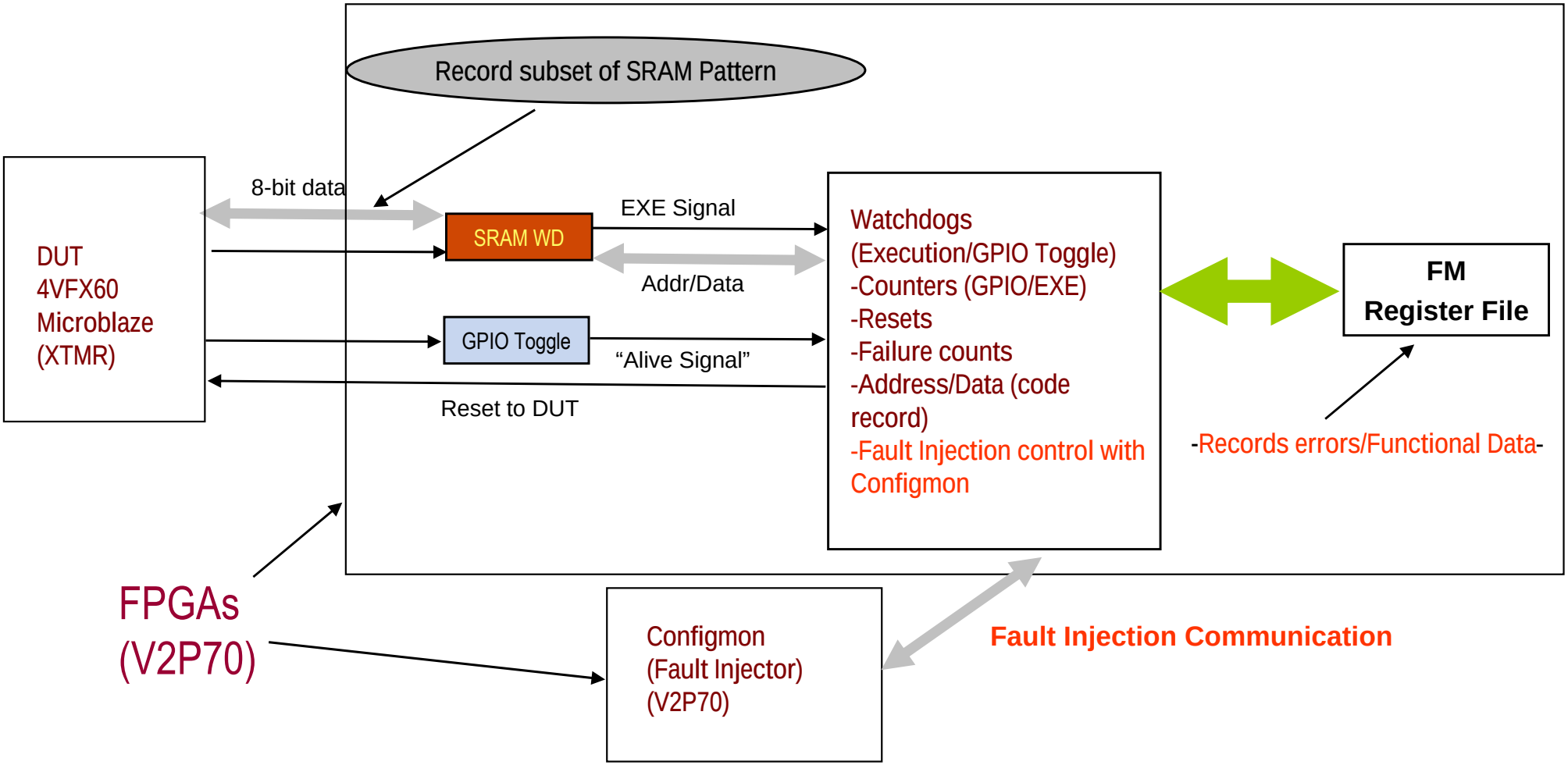
DUT Block Diagram

- Uses XTMR (Xilinx Triple Modular Redundancy) Flow
- Scrubber Block RAM wrappers for each type of BRAM



Functional Block (simplified) Diagram

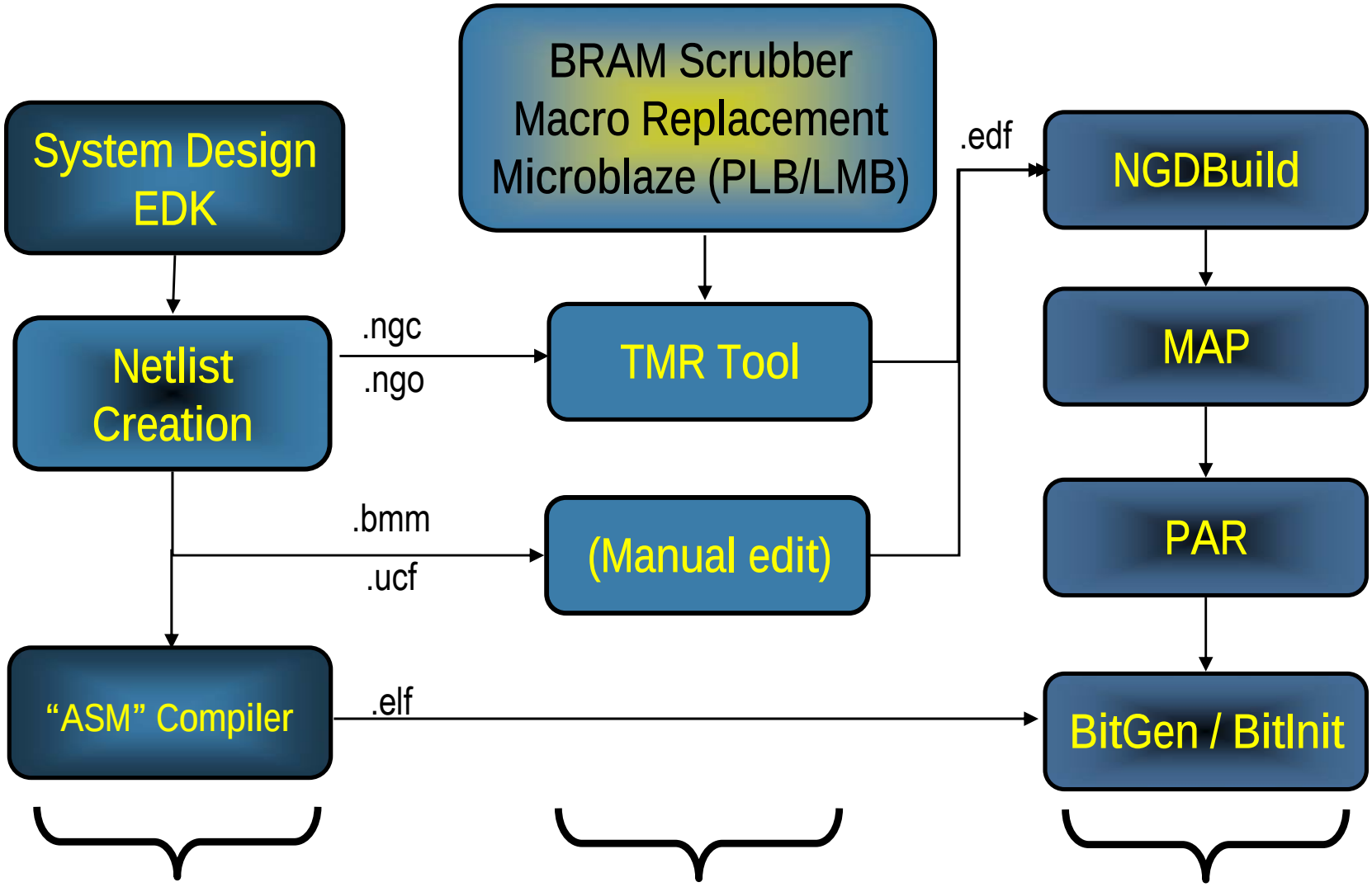
- Code in DUT Block RAM (For this test) – Assembly only
- Watchdogs (Memory Test pattern Pass/Fail) - optional
- Control of Fault Injection Engine (Configmon)



Mitigation Flow Overview

- Use the XTMR (Xilinx Triple Modular Redundancy) flow techniques to mitigate the MicroBlaze Soft logic
  - MicroBlaze designs consist of the bus structure (PLB) – Triplicated
  - MicroBlaze Core - Triplicated
  - Surrounding peripherals such as UART, SRAM Controller, GPIO, etc… - Triplicated
  - All IO - Triplicated
  - “BRAM scrubber” on PLB bus BRAM/DLMB

Overall EDK / TMRTool Design Flow



Processor Fault Injection Overview

Controlled by Funcmon

➢Coordination of Failures before next bit

➢Must pass several "Toggles" before declaring a pass (This test - 16)

- Used to inject configuration faults to test mitigated strength of the Soft Processor
- Configuration Monitor can take commands from Funcmon to coordinate bit upset locations
  - Any monitoring of functionality is done via a *Functional Monitor* FPGA
  - Funcmon sends commands to fault injector for maximum control
    - Configmon can then record bit upset locations
      - *Block RAM content can not be scrubbed via the configuration port* – use Block RAM scrubber (*BRAM content not affected by Fault injection*)
      - See XAPP962 "Single Event Upset Mitigation for Xilinx FPGA Block Memories"
  - Fault inject the FX60 in ~60minutes (depends on number of toggles to declare a pass)
  - "Back Annotate" Failure bits for further analysis (gives net name up failing bits)
  - Simple ASM design (no "C"). Eliminates corruption of Code via a write
  - RAM is "Read Only" (for this simplified test)

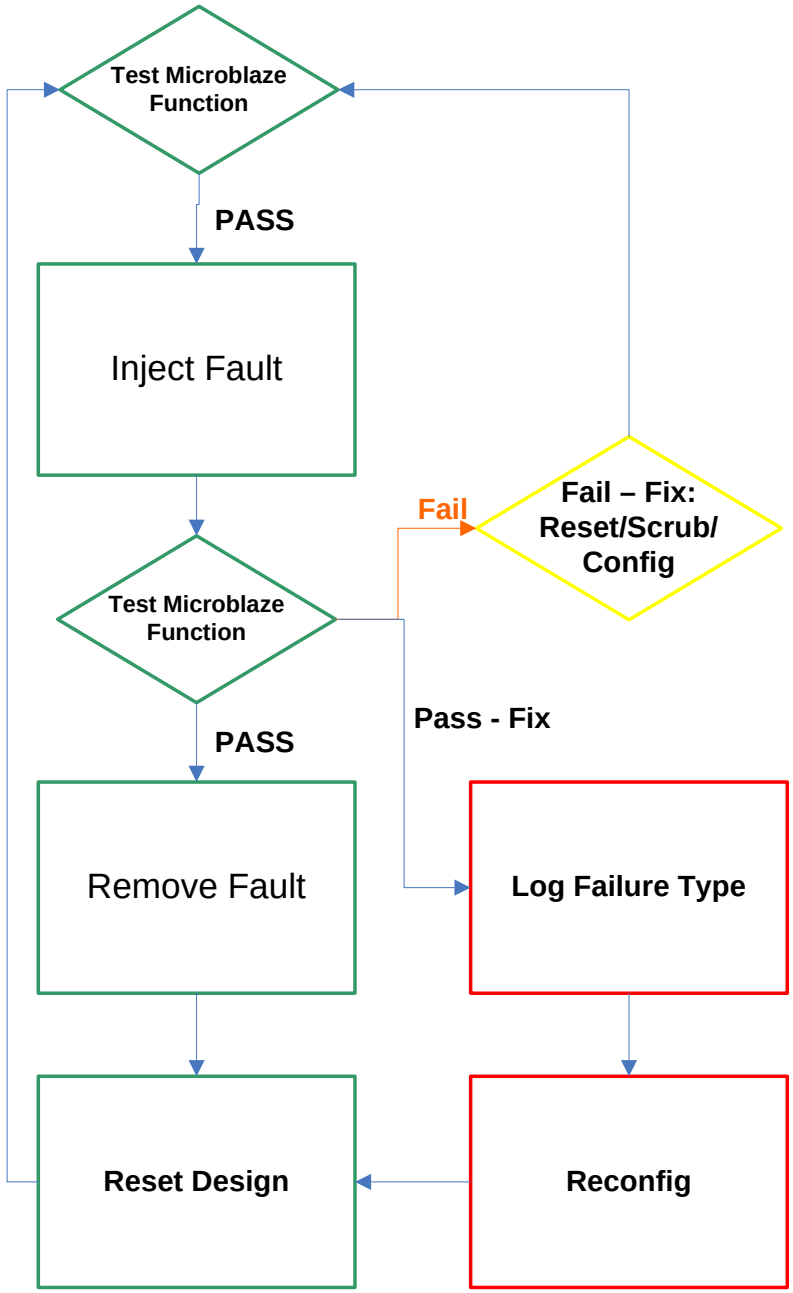
Fault Injection : Types Of Errors

(Each type of error seen after a single fault)

- Types of errors Recorded/Corrected
  (*Bigger hammer each time*)

  1). Reset Only – Reset “fixes” processor
  2). Scrub + reset – (A scrub of the faulty bit fixes processor)
  3). Reconfigure – The part must be reconfigured
  4). “Re-Load of RAM” – Eliminated in this test with “read Only” (will need to re-visit)

Funcmon High-Level Fault Injection Flow (Simplified)



Fault Injection Results

Tool/Type	Slices(Total)	SEFIs	FM Rst Bits	FM Scrub Bits	FM CFG Bits	FM4 Bits	Total FM Failures (SEFIs not counted)	Notes/Back Annotate
Single String(Ext)	2120	9	1027	27060	228	0	28315	ASM -> External RAM execution (Simple Funcmon – reset after each bit)
XTMR (External RAM)	8958	9	807	288	4	0	1099	ASM -> Internal RAM execution (Simple Funcmon – reset after each bit)
Single String(Int RAM)	2778	9	1214	22047	887	0	24148	Simple ASM Program Only (Internal RAM) wait after each FI. Reconfig after each error. (Isolation of SPF bits)
XTMR (Internal) No Area Group	11224	9	2	6	1	0	9	FM -> scrub->wait after each FI. Reconfig after each error. (Isolation of SPF bits)
XTMR (Area Group)	11224	9	0	6	2	0	8	FM -> scrub->wait after each FI. Reconfig after each error. (Isolation of SPF bits)
XTMR (Area Group) (SRL16s removed – Synthesis Script)	23616	9	0	4	0	0	4	FM -> scrub->wait after each FI. Reconfig after each error. (Isolation of SPF bits) – EDK Synthesis settings/SRL16 removed

Tips for high reliability mitigation

- Triplicate all logic
- Triplicate IO
  - Separate each domain into different banks
    - Eliminates IO “SEFI”
- Use “more robust” synthesis settings in EDK (not default) – *script needed.*
- Use a BRAM scrubber
- Turn off Global Optimization in MAP

Fault inject the design

- Use fault injection (on hardware) to simulate and troubleshoot potential problems before beam testing

Challenges

- Had to move to a “ROM” with Assembly structure to avoid writes (for now)
  - Does not allow for full reset as RAMs may get corrupted
  - Will look into workaround
  - Allows focus on the MicroBlaze core only
- Only partial reset
  - May be due to SRL16s/Distributed RAMs (see next bullet)
  - Fix any errors by reconfigure (for bit isolation)
- Bits that fail in the Fault injector may be due to “previous bits” together
  - Need to add a recording feature to go back and isolate multi failures and associate bits with each other.
- External RAM – needs to be re-implemented and tested
  - Move the IOs to different banks
- These tests were for Fault Injection bit studies only.
  - Not a “real life” test
  - However, does give insight into failures seen in beam
  - Will implement beam “simulator”

Lessons Learned

- Do a single Version of the design first to work out the hardware/software bugs
- Use a netlist viewer to inspect the triplicated design to make sure it is constructed as expected
- Remove ChipScope
- Remove DCMs – put mitigation back in later
- Use Fault Injection testing before beam testing
- Separate Triplicated IOs into different banks (to do next)
- Use a better test than a single bit toggle (implement later)
- Use “Bit Isolation” to isolate actual failing bits (re-sync design)
- Past bit may cause failures
- After an injection, the circuit needs time to “recover” – vote out the failures in loops
- Area groups in constraint file, may help
- Turn off Global Optimization in MAP – Can remove voters
- Turn off Clock Enables in Synthesis if possible
- Change Synthesis settings in EDK (With script) i.e. better state machine synthesis
- Use simple ASM program for now. “C” may cause issues as the vector tables are re-written on boot up and may cause problems. Looking into this.
- With bit injected, use “smoke test” (knock out a domain) to determine domain reliance failure
- Running the same design can cause different bit failures
  - Run the same design and compare reports

Conclusion/Future work

- Mitigation of a soft processor in a complex FPGA is a viable solution
  - Still working out remaining mitigated faults
- Care must be taken to mitigate properly
  - Testing (Fault injection)
  - Proper tool flow must be understood

Future work :

- Still refining Virtex 4 Soft processor mitigation techniques
  - Refine flow for all users
- Publish design flow
- Add more complexity to system tests
- Make Fault Injection more reliable and consistent
- "Simulate Beam" with scripting (Upsets/scrub cycle)

- External memory (More testing) – SRAM/SDRAM/DDR
  - Look into Write Enable RAM corruption
- Cache/DCMs etc… (more complexity)

References

1. "Robust FPGA/Embedded-Processor Design: Design Flow for SEU Mitigation" - Greg Miller, Carl Carmichael, Gary Swift, MAPLD 2006
2. **Rezgui S., "SEU Mitigation of a Soft Embedded Processor in the Virtex-II FPGAs", September 2005**
3. "A Test Methodology for Determining Space-Readiness of Xilinx SRAM-based FPGA Designs" Heather Quinn, Paul Graham, Keith Morgan, Michael Caffrey, and Jim Krone
4. Upset Susceptibility and Design Mitigation of PowerPC405 Processors Embedded in Virtex-II Pro FPGAs" – Swift/Allen/George/Rezgui/Carmichael/Chayab – MAPLD 173 (2005)
5. "Upset Characterization of the PowerPC405 Hard-Core Processor Embedded in Xilinx Virtex-II Pro Field Programmable Gate Arrays" – Swift/Petrick/Chayab/George/Allen/Farmanesh - NSREC Data Workshop (2006)
6. "An Upset-Mitigated FPGA-based High Performance Compute Platform for Space Applications" – MAPLD 177 (2006) – Swift/Allen and SEU Consortium
7. **Lima, F., Carmichael, C., Fabula, J., Padovani, R. and Reis, R., "A Fault Injection Analysis of Virtex® FPGA TMR Design Methodology", *RADECS'01,* September 2001.**
8. **Carmichael C., "Triple Module Redundancy Design Techniques for Virtex FPGAs", *http://www.xilinx.com/bvdocs/appnotes/xapp197.pdf*, Xilinx Application Note XAPP197, November 2001.**
9. **TMR Tool User Guide, UG156, Version 6.2.3, *http://support.xilinx.com/products/milaero/ug156.pdf*, Xilinx Inc., September 2004.**

Additional Information

BRAM Mitigation Methodology

- Apply TMR on the used BRAMs
- Create a BRAM scrubber macro (to replace a single port BRAM)
- Determine BRAM replacement locations in TMRTool

- Each Block RAM primitive collection (may contain several primitives) is replaced with the Block RAM scrubber macro.
- Two types of Internal BRAM used
  - LMB (Local Memory Bus)
  - PLB (Hangs off of the PLB Bus)

Setup/Fault Injection Test types

Design Setup :

-All code is running from the PLB Block RAM

-Simple ASM code to control toggle of GPIO bit (no writes allowed)

-Reset After every injected fault – Simple Funcmon

-Scrub, reset, wait after each fault injection, reconfig after each error (bit isolation)

NOTE : Not realistic for beam testing, but useful to determine single points of failure.

16 passes (toggles) are required for a pass (For each bit)

Test Types shown in this presentation:

1). Single String – Non triplicated with simple reset after FI

2). XTMR – External RAM with simple reset after FI

3). Single String Internal RAM– With FI and scrub after every bit/Reconfig after every Failure

4). XTMR Internal RAM (no AG) - With FI and scrub after every bit/Reconfig after every Failure

5). XTMR Internal RAM (AG) - With FI and scrub after every bit/Reconfig after every Failure