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

Greg Miller¹, Carl Carmichael¹, Gary Swift¹, Chen Wei Tseng¹

MAPLD 2008

¹Xilinx, 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

FPGA
(4VFX60 Xilinx Tested)

• JTAG
  “Debug”

• MDM

• Microblaze/PPC

• UART-Lite

• RS232

• BRAM

• BRAM scrubber

• GPIO monitor in FM

• GPIO Toggle Test

• SRAM messages

• Heartbeat

• PLB BUS (A.K.A “XPS” Bus)

• Executable Code

• “PLB” Memory BRAM scrubber

• SRAM (Emulation)

Dev Board

 Miller 3 MAPLD 2008/Session D
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)

DUT
4VFX60 Microblaze (XTMR)

FPGAs
(V2P70)

Fault Injection Communication

Record subset of SRAM Pattern

SRAM WD

8-bit data

Addr/Data

EXE Signal

“Alive Signal”

Reset to DUT

GPIO Toggle

Watchdogs (Execution/GPIO Toggle)
- Counters (GPIO/EXE)
- Resets
- Failure counts
- Address/Data (code record)
- Fault Injection control with Configmon

Configmon (Fault Injector) (V2P70)

FM Register File

- Records errors/Functional Data-
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

System Design
EDK

Netlist
Creation

“ASM” Compiler

BRAM Scrubber
Macro Replacement
Microblaze (PLB/LMB)

TMR Tool

NGDBuild

MAP

PAR

BitGen / BitInit

Design Entry
EDK/ISE (9.2i)

XTMR Conversion
TMR Tool (9.2i)

Implementation
ISE (9.2i)

Implementation
ISE (9.2i)
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)

1. Test Microblaze Function
   - PASS
   - Inject Fault
     - Fail
       - Fail – Fix: Reset/Scrub/Config
     - Pass
       - Pass - Fix
       - Log Failure Type
         - Reconfig
         - Reset Design
           - Remove Fault
             - Test Microblaze Function
               - PASS
## Fault Injection Results

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

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