Rapid Prototyping of Radiation-Tolerant Embedded Systems on FPGA

F.Restrepo-Calle\textsuperscript{1} A.Martínez-Álvarez\textsuperscript{1} F.R.Palomo\textsuperscript{2} H.Guzmán-Miranda\textsuperscript{2} M.A.Aguirre\textsuperscript{2} S.Cuenca-Asensi\textsuperscript{1}

\textsuperscript{1}Computer Technology Department, University of Alicante, Spain
\textsuperscript{2}Department of Electrical Engineering, University of Sevilla, Spain

FPL 2010
August 31 - September 2, 2010
Milano, Italy
1 **Introduction**
   - Problem
   - Solutions for Embedded Systems
   - Objectives
   - Fault Model and Terminology

2 **Platform for the rapid prototyping of dependable ES**
   - Platform for fault-tolerant co-design
   - Software Hardening Environment
   - Reliability Evaluation Tool

3 **Case Study**
   - Picoblaze
   - Software-based technique: SWIFT-R
   - Hardware-based technique: TMR
   - Prototypes evaluation

4 **Conclusions and Future Work**
Problem

Miniaturization of electronic components

〜〜 **Advantage:** microprocessor performance increase
〜〜 **Disadvantage:** more prone to *transient faults* [Baumann, 2005]

Transient faults (soft-errors)

- Induced by radiation - by the ionization of an incident charged particle
- Do not cause permanent damage
- Can alter signal transfers or stored values provoking errors in the systems
- Occur in: space, atmosphere and even at ground level [Baumann, 2002]

〜〜 Increasing concern about the fault mitigation in mission-critical, security and safety systems
Solutions for Embedded Systems

**Hardware-based techniques**

- Usual solution: hardware redundancy
- Low level structures: ECC, parity bits, TMR
- More complex components: co-processors
  
  - [Mahmood and McCluskey, 1988], *functional units* [Austin, 1999], . . .

- Exploiting redundancy in multi-thread/multi-core architectures
  
  - [Gomaa et al., 2003, Mukherjee et al., 2002]

- **Advantage**: high effective solution
- **Disadvantage**: very costly!
In recent years several proposals based on redundant software

Some examples: *EDDI* [Oh et al., 2002b], *CFCSS* [Oh et al., 2002a], *SWIFT* [Reis et al., 2005b], *ARBT* [Rebaudengo et al., 2001], . . .

**Advantage**: low cost with acceptable reliability

**Disadvantages**: increment code size and execution time
Solutions for Embedded Systems

Hybrid Hardware/Software approaches

- In many cases the optimal solution is an intermediate point, which combines software and hardware protection approaches  
  ⇒ **HW/SW fault-tolerant co-design**

- Examples: [Bernardi et al., 2006, Reis et al., 2005a]

- Need for suitable tools to easily explore the design space in order to find the best trade-off between design constraints and reliability requirements

- Growing use of FPGAs to prototype ASICs as part of an ASIC verification methodology
So in this paper, we present...

A rapid prototyping approach for radiation-tolerant embedded systems design

- FPGAs are used as development and verification platform to produce HW/SW systems that best meet the design and dependability constraints
- Mitigation techniques are applied to a high abstraction level so the final deployment platform will be an ASIC or an FPGA
- Supported by a hardening platform that is made up of:
  - Software Development Environment: to implement, automatically apply and evaluate software-only fault tolerant techniques
  - FT-Unshades: FPGA-based fault emulation tool to assess several reliability metrics
Fault Model: Single Event Upset — SEU [Rebaudengo et al., 2001]

- Only one bit-flip occurs in a storage cell during the program execution
- Widely used because matches the real fault behavior

Faults classification [Mukherjee et al., 2002]

According to their effect on the program behavior:

- **unACE**: fault in a unACE bit, i.e. the program finishes and produces the expected results
- **SDC**: Silent Data Corruption - finishes with incorrect results
- **Hang**: abnormal program termination or infinite loop
Platform for fault-tolerant co-design

**HW/SW Fault-Tolerant Co-design**

- **Designer**
  - Design constraints
  - Reliability requirements
- **SW Design Tools**
- **HW Design Tools**
- **Reliability Evaluation Tools**
Platform for fault-tolerant co-design

HW/SW Fault-Tolerant Co-design - Proposed tools

- Designer
  - Design constraints
  - Reliability requirements

- Software Hardening Environment
- HW Design Tools
- FT-Unshades

Our tools
Third-party tools
Software Hardening Environment

General scheme

Compiler front-ends

Original source code

Arch. 1
Arch. 2
...
Arch. n-1
Arch. n

Generic Instruction Flow (GIF)

Generic Hardening Core (GH-Core)

Hardener
Simulator

Hardened Generic Instruction Flow (HGIF)

Compiler back-ends

Arch. 1
Arch. 2
...
Arch. n-1
Arch. n

Hardened source code
Software Hardening Environment

Advantages

Main advantages

- Based on a Generic Architecture permits...
  - to handle multiple microprocessors
  - to provide an uniform hardening core
  - to re-target the output to any supported microprocessor
- Automatic code transformation based on rules (assembler)
- Conceived to implement a wide suite of techniques
Software Hardening Environment

Generic Architecture

Three main topics:

- **Generic Instruction**

<table>
<thead>
<tr>
<th>Address</th>
<th>Mnemonic</th>
<th>Generic Operator List</th>
<th>Affected Generic Flag List</th>
<th>Instruction Type</th>
<th>Tool message</th>
</tr>
</thead>
</table>

- **Memory Management**
  - Identification of memory map (and memory sections)
  - Update memory map when code is inserted:
    - Dilation
    - Displacement
    - Reallocation

- **Control Flow Graph**

Node 1: {I1, I2, I3, I4, I5}
Node 2: {I6, I7, I8}
Node 3: {I9, I10}
Node 4: {I11, I12, I13}
Node 5: {I14}
Sphere of Replication — SoR [Reinhardt and Mukherjee, 2000]

- Logic domain of redundant execution
- Instruction classification (for hardening):
  - InSoR
  - OutSoR

we have applied this concept in a flexible way \(\sim\) a SoR with flexible frontiers facilitates implement selective software protection
Nodes and Subnodes

Every node is subdivided into subnodes after each *OutSoR* instruction

Node 1

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>I1:</td>
<td>______</td>
<td>I2:</td>
<td>______</td>
<td>I3: STORE</td>
</tr>
<tr>
<td>I4:</td>
<td>______</td>
<td>I5:</td>
<td>______</td>
<td></td>
</tr>
</tbody>
</table>

Subnode 1

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>I1:</td>
<td>______</td>
<td>I2:</td>
<td>______</td>
</tr>
<tr>
<td>I4:</td>
<td>______</td>
<td>I5:</td>
<td>______</td>
</tr>
</tbody>
</table>

Subnode 2
Software Hardening Environment

Generic Hardening Core

**Generic Hardening Core: Hardener**

Two components: *Hardener* and *Instruction Set Simulator*

*Hardener*

⇝ Tool for the design of software-based techniques:

- API of hardening routines
- Flexible and easy to extend

⇝ Allows to automatically apply these techniques:

- Receives a GIF and produces the Hardened-GIF
- User control from command-line

**Options:** method, mcpu, redundancy level, replication times, voter, ...
Software Hardening Environment

Generic Hardening Core

Generic Hardening Core: ISS

*Instruction Set Simulator — ISS*

- Simulates the execution of the *GIF*
- Checks original and hardened code functionality - custom *pragmas* with the expected results
- Outputs useful information (code and execution time overheads, program characterization, . . .)
- Simulates *SEU* faults to preliminary evaluate the reliability
Reliability Evaluation Tool: FT-Unshades

[Guzmán-Miranda et al., 2009, Napoles et al., 2007]

- FPGA-based platform for the reliability evaluation
- Emulated bit-flips in the real implementation of the system by means of partial reconfiguration
- Smart Table: \textit{FT-Unshades} extension for the study of microprocessor architectures
Reliability Evaluation Tool

Reliability Evaluation Tool: FT-Unshades

...is used by the European Space Agency — ESA
Case study: Picoblaze

- Compiler front-end and back-end for *Picoblaze*
- *Picoblaze*: 8-bit soft-microprocessor widely used in FPGA-based systems
  - Strong restrictions on memory program size and performance
- *RTL Picoblaze* developed
## Benchmark suite

- Bubble sort (*bub*)
- Scalar division (*div*)
- Fibonacci (*fib*)
- Greatest common divisor (*gcd*)
- Matrix addition (*madd*)
- Scalar multiplication (*mult*)
- Matrix multiplication (*mmult*)
- Exponentiation (*pow*)
Software-based technique: SWIFT-R

SWIFT-R

Software protection: SWIFT-R [Reis et al., 2007]

TMR-based method aimed to recover faults from the data section

- Build the control flow graph
- Data triplication after \textit{inSoR} instructions
- Triplication of instructions using redundant data
- Insertion of majority voters and recovery procedures before \textit{outSoR} instructions and before conditional branches
Software-based technique: SWIFT-R

### Code and Execution Time Overheads

<table>
<thead>
<tr>
<th>Normalized Overheads</th>
<th>bub</th>
<th>div</th>
<th>fib</th>
<th>gcd</th>
<th>madd</th>
<th>mmult</th>
<th>mult</th>
<th>pow</th>
<th>GeoMean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Overhead</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execution Time Overhead</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Hardware protection: **TMR**

- The fault tolerant co-design strategy was complemented by incrementally hardening the microprocessor resources.
- Five microprocessor versions were developed:
  - **P0**: non-hardened *RTL Picoblaze*
  - **P1**: with hardware redundancy for Program Counter (*PC*), Flags and Stack Pointer (*SP*)
  - **P2**: all registers in the pipeline protected
  - **P3**: with hardware redundancy for *PC*, Flags, *SP*, and Pipeline
  - **P4**: full protected
Reliability evaluation: Experimental Setup for FT-Unshades

- For each prototype: Fault injection campaign with selective attacks against the microprocessor register sets: register file, $PC$, $Flags$, $SP$, and pipeline.
- For each one of these register sets, 5,000 SEUs injected (one per execution).
- Bit-flip in a randomly selected clock cycle.
Fault classification percentages for every test program —non-hardened (O) and SWIFT-R (H)— running on each processor version (P0 to P3)
Prototypes evaluation

Reliability vs Hardware Costs

Normalized hardware cost and percentage of unACE faults per microprocessor
Conclusions and Future Work

- We have presented a rapid prototyping approach for the design of radiation-tolerant embedded systems using FPGA.
- This approach is supported by a flexible hardening platform, which facilitates the representation of several trade-offs among design constraints, reliability, performance, and costs.
- The rapid prototyping strategy allows designers to easily explore the design space between hardware-only and software-only fault-tolerance techniques.
- As case study, several fault-tolerant prototypes based on a RTL implementation of PicoBlaze have been developed and evaluated.
- The infrastructure will be extended to support 32-bit microprocessors, such as: Microblaze and Leon3.
Thank you for your attention!

September 1st 2010, FPL 2010, Milano, Italy


Concurrent error-detection using watchdog processors - a survey.


Control-flow checking by software signatures.  
*IEEE Transactions on Reliability, 51*(1).

Error detection by duplicated instructions in super-scalar processors.  
*IEEE Transactions on Reliability, 51*(1).

A source-to-source compiler for generating dependable software.  

Transient fault detection via simultaneous multithreading.  
Conclusions and Future Work


