A High-Level Tool for Bit-level SEU Sensitivity Analysis of DSP Filters

Andrew Mast, Jamie Montealegre, Luke Jenkins, and Srinivas Katkoori*
Computer Science and Engg., University of South Florida, Tampa, FL

Andrew White and Cliff Kimmery
Space Electronic Systems Div., Honeywell Inc, Clearwater, FL

*Contact Author: katkoori@cse.usf.edu
Overview

• Motivation
• Problem Formulation
• Proposed Approach
• Implementation
• Experimental Results
• Conclusions and Future Work
Space Electronic Systems – Radiation Effects

• Space Electronic Systems
  – Radiation effects can be fatal to the mission

• System hardening by redundancy
  – Triple Modular Redundancy (TMR)
    • Spatial TMR incurs 200% area overhead
    • Temporal TMR incurs 200% performance overhead

• Redundancy is too costly!!
  – Can we do better?
  – Can we exploit the function properties?
Single Event Upset (SEU)

- A momentary flip in a bit value due to radiation
- If latched can become permanent

```
00000
↓  ← Radiation
00010
↑
Flip in data value
```
Digital Signal Processing (DSP) Filters

- DSP Filters widely used in space missions widely
  - noise removal
  - tuning to frequency ranges of interest
  - signal extraction, etc

- Representation
  - Data Flow Graph

- Filter characteristics
  - Frequency Response
  - Normalized (typically)
Problem Formulation

Given a DSP Filter, identify the SEU sensitive nodes of the Filter
- Such sensitive nodes can then be hardened
Scope of this work

• **Functionality**: Limited to DSP FIR Filters
• **Radiation Effects**: Single Event Upsets
• **Target Architecture**: None
  • Early analysis tool
Proposed Approach

Step 1: Examine the effects of an SEU on a filter.

Step 2: Compare these effects with normal filter behavior.
Proposed Approach

Step 1: Examine the effects of an SEU on a filter.
  • How do we...
    • Represent a filter?
    • Model an SEU on a filter?
    • Examine the effects of an SEU on a filter?

Step 2: Compare these effects with normal filter behavior.
  • How do we compare behavior?
Step 1: SEU Effect on DSP Filter

- Represent a filter with a data flow graph.

- Model an SEU by flipping edge bit values.

An SEU occurred on the LSB of this edge

Binary
4'b - 1010
4'b - 1011
Step 2: Comparing Frequency Response

- Compare filter behavior by using root mean square error on amplitude responses.

![Amplitude Responses Diagram]

- A Filter affected by an SEU
- A Normal Filter
Analyzing a DSP Filter with an SEU

Input
Pure Tones

FIR FILTER
SEU On Edge/Bit

Fourier Transform

FFT Output
SEU Sensitivity Analysis

SEU Analyze(Filter, Frequency Range)
begin
  Normal_Response = FrequencyResponse(Filter, FrequencyRange);
  foreach edge in the DFG
    foreach bit in the edge
      SetSEU(edge, bit)
      SEU_Response = FrequencyResponse(Filter, edge, bit, Frequency Range)
      SensitivityList[edge, bit] =
        CompareFrequencyResponses(Normal_Response, SEU_Response)
      ClearSEU(edge, bit)
    end foreach
  end foreach
return SensitivityList
end algorithm
Software Functions

• “Simulating” A Filter with a DFG
  • AIF File
  • Coefficient File
  • Sine Wave Generation
  • Generate the frequency response and compare the result with Matlab’s Filter Designer Tool

• SEU Function
  • Store edge values as an integer variable
  • Add or subtract $2^k$ from edge value.
SEU Function

- Introduces an SEU on a given bit of a given edge
  - Example of concept: edge = 3; bit = 2;
    - Current value of edge = 0101
    - Shift
      - 0001
    - Check if LSB is even or odd
    - Flip the bit by adding or subtracting $2^{\text{bit}}$ to original value

```c
// SEU function
m = EdgeValue / (int)2^bit;
if (m % 2 == 0) //If m is even
    EdgeValue = EdgeValue + (int)2^bit;
else //else m is odd
    EdgeValue = EdgeValue - (int)2^bit;
//Reset edge and bit to null
edge = null;
bit = null;
```
Software Implementation

• Created a Java based program that reads in:
  – .aif files describing the DFG Structure
  – .txt files containing coefficient list

• Resources:
  – Michael Thomas Flanagan's FFT Java Scientific Library**
    – Non-Commercial use only!
    – We recommend JMSL Numeric Library for commercial operation.
  – Java.Math Library

• Program outputs a text file containing:
  – Amplitude Response
  – RMSE values

## Experimental Validation

<table>
<thead>
<tr>
<th>Filter Type</th>
<th>Low Range</th>
<th>Medium Range</th>
<th>High Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Band Pass</td>
<td>10 – 20 KHz</td>
<td>100 – 200 KHz</td>
<td>1 – 2 MHz</td>
</tr>
<tr>
<td>High Pass</td>
<td>10 KHz</td>
<td>100 KHz</td>
<td>1 MHz</td>
</tr>
<tr>
<td>Low Pass</td>
<td>10 KHz</td>
<td>100 KHz</td>
<td>1 MHz</td>
</tr>
</tbody>
</table>

- **Low pass test #1.**
  Pass-band from 0 to 0.2

- **Low pass test #2**
  Pass-band from 0 to 0.3

- **Low pass test #3.**
  Pass-band from 0 to 0.7
Effect of an SEU on Amplitude Response
3D Error Plots

%Error

Primary Inputs...... Intermediate...... Primary Outputs

DFG Edges

LSB...... MSB

Bits
Test Results: Low Pass Filter

Low Frequency

Medium Frequency

High Frequency
Test Results: High Pass Filter

Low Frequency

Medium Frequency

High Frequency
Test Results: Band Pass Filter

- Low Frequency
- Medium Frequency
- High Frequency
## RMSE vs. Overhead

<table>
<thead>
<tr>
<th>RMSE vs. Triple Modular Redundancy</th>
<th>Highpass Filter, Total Edges (483)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMSE</td>
<td>≤ 1</td>
</tr>
<tr>
<td># of Edges to Protect</td>
<td>418</td>
</tr>
<tr>
<td>Redundancy %</td>
<td>173%</td>
</tr>
</tbody>
</table>

- 200% Redundancy is unnecessary for our test filters.
- High RMSE values can still result in acceptable filter behavior.
SEU Analysis Tool – Implementation Summary

• **Automatic Filter Simulation** Software that simulates a DSP filter via a DFG.
  – Accept a data flow graph representation of DSP filter as input in AUDI Intermediate Format (AIF).
  – Provide correct output based on the test vector and filter in a format defined by the team.

• **Automatic Response Extraction** Effects of SEU on Filters Performance

• **Extensibility** The software will be designed to be extensible by creating separate modules for input, simulation, and SEU sensitivity analysis.

• **Extensive Documentation - Users and Developers.**
  – User documentation should include instructions on how to input the DFG, run the program, and interpret the output.
  – Developer documentation should include a list of classes and a description of their methods and properties, as well as an explanation on how and why they were designed.
Conclusions

- For DSP FIR Filters, full TMR may not be necessary.
- Overhead due to redundancy is a function of error tolerance on Filter Performance.
Future Work

• **Program Extension**
  – Explore response comparison methods
    • Characteristic Comparison
    • Weighted
  – Phase Response

• **Test on different Filter forms**
  – Are some DSP Filter structures are better than others?

• **Batch support**
References


