Safe and Efficient One-Hot State Machine

Sunant Katanyoutanant, Martin Le, Jason Zheng

Jet Propulsion Laboratory
Introduction

- Finite State Machines (FSM) are important building blocks of avionic control systems.
- Single Event Upsets (SEU) can corrupt FSM state registers
- Substantial research done in making FSM safer
- Asynchronous Nature of SEU complicates mitigation mechanisms
Goals/Roadmap

1. How do asynchronous SEU affect FSM logic?
2. How to mitigate the effects of SEU?
3. Based on 1 and 2, what improvements can we make to the existing mitigation methods?
4. How to implement the improvements?
Overview

• **Background Information**
• Asynchronous SEU Analysis (Question 1+2)
• Existing Mitigation Methods
• Challenge
• One-Hot with Pipelined XNOR (Question 3)
• Python Script Automation (Question 4)
• Test Results
• Future Outlook
Background - FSM

- Set of States \( S_1 \ldots S_n \)
- Set of Inputs \( I_1 \ldots I_m \)
- Set of Outputs \( O_1 \ldots O_k \)
- Output Function (OF)
- Next State Function (NS)
Background – State Encoding

- Each state $S_i$ is represented by a binary pattern $P_i$, where $i$ is an arbitrary index.
- A mapping from the state index $i$ to $P_i$ is the state encoding function $E$.
- Binary (sequential) encoding: $E(i) = i$
- One-hot encoding: $E(i) = 2^i$
- Others: grey, johnson, hamming-2, etc.
Overview

• Background Information
• **Asynchronous SEU Analysis**
• Existing Mitigation Methods
• Challenge
• One-Hot with Pipelined XNOR
• Python Script Automation
• Test Results
• Future Outlook
SEU – First Look

- Single Event Upsets are radiation-induced errors caused by passing charged particles
- SEU is asynchronous
- Two effects of SEU:
  - Logic glitch: temporary
  - Bit-flip: persistent until next clock
    - Error state: an erroneous but defined state
    - Illegal state: an undefined state (FSM stuck)
Logic Glitches

- **Green Zone**: logic glitches can be waited out
- **Red Zone**: undeterministic (distance to D, and timing)
  - Erroneous result
  - Meta-stable result
  - Correct result
Bit-flips w/o Detection/Correction

- SEU occur in red zone: undeterministic next state
  - Error propagates through some or all paths – erroneous or illegal
  - Meta-stable
  - Correct state
- Green: definitely erroneous or illegal next state
Bit-flips w/ Detection/Correction

- SEU occur in red zone: undeterministic **next state**
  - No detection or correction
  - Partial detection or correction
  - Meta-stable
- Green = (likely) detectable or fixable error
Shrinking the Red Zone

• Improves the chance of detecting and/or correcting an SEU
• More deterministic
• How?
  – Cannot change setup time
  – Often cannot change clock frequency
  – Only thing left is to make FSM faster
• Faster = Safer!
How to Design Faster FSM?

• Speed up output logic (usually not the bottleneck)
• Speed up next-state logic
• Speed up error detection/correction
Overview

• Background Information
• Asynchronous SEU Analysis
• Existing Mitigation Methods
• Challenge
• One-Hot with Pipelined XNOR
• Python Script Automation
• Test Results
• Future Outlook
Existing Mitigation Methods

• SEU Correction
  – Triple Modular Redundancy (TMR)
  – Hamming-3 Companion States

• SEU Detection
  – The “default” case (full-state lookup)
  – Hamming-2 (binary + parity)
  – One-hot + XNOR tree
Triple Modular Redundancy

- Simple concept
- Tried-and-hard
- Applicable to any sequential logic
- Slow (large red zone)
- Vulnerable to glitches
Hamming-3 Companion States

• Every legal state has $m$ companion states, where $m = \text{width of state register}$.
• $n \cdot (m+1) \leq 2^m$ (compare to $n \leq 2^m$), where $n = \text{number of legal states}$
• One bit-flip turns a legal state into a companion state
• Corrects one bit of error by looking up the legal states and all possible companion states
• A 16-state FSM needs 112 companion states

Zheng
Page 17
MAPLD2005/179
Companion State Example

Total number of states:

\[2^7 = 128\]

Number of legal states:

\[2^7 / (7+1) = 16\]
The “Default” Case

always @ *
case (state)
    st1: nextState = …
    st2: nextState = …
    …
    default: nextState = …
    …
endcase

Zheng
The “Default” Case (cont.)

• Cannot detect error when SEU flips state registers to another defined state
• Big performance hit and area overhead with sparse encoding
• Some logic synthesizers ignore default states to optimize FSM
Hamming-2 (Binary + Parity)

- Minimum hamming distance of 2
- $E(n) = n + \text{parity}(n)$
- Can detect single-bit error by XORing all the bits
- Slightly slower than binary encoding but more robust
- Tested and preferred by JPL Avionics Electronics Section

Example H-2 Code:

00000
00011
00101
00110
01001
...
11110
One-hot + XNOR

- 1-bit State Lookup
- Sparse code
- Use multi-input XNOR logic to detect illegal states
- XNOR can be tricky to implement
- XNOR Logic levels = \( \log_2 N - 1 \)
How to Design XNOR Logic

- \( \text{XNOR}(A_1, A_2, A_3, \ldots, A_n) \neq A_1 \sim^\wedge A_2 \sim^\wedge A_3 \ldots \sim^\wedge A_n \)
- No easy or direct way to describe XNOR in HDL
- Most synthesis tools cannot generate fast and well-balanced XNOR tree
- For best results, must manually write the HDL code
- For FPGA with 4-LUTs, need \( \log_2(N)-1 \) levels of logic, where \( N \) is the number of inputs.
- Must use synthesis directives to prevent logic pruning
A 16-Input XNOR Example

- F is a 4-bit adder with some modifications
- G is a 4-input XNOR
- 3-level logic tree
- Works well for 4-LUT (Look-Up Table) architectures
- FSM logic levels $\geq 4$
Overview

• Background Information
• Asynchronous SEU Analysis
• Existing Mitigation Methods
• **Challenge**
• One-Hot with Pipelined XNOR
• Python Script Automation
• Test Results
• Future Outlook
Can We Do Better?

• What is “better?”
  – More deterministic under asynchronous SEU
  – Faster logic (smaller red zone)
  – Small area overhead
  – Easy to code

• Our approach: one-hot with pipelined XNOR detection
Overview

• Background Information
• Asynchronous SEU Analysis
• Existing Mitigation Methods
• Challenge
• One-Hot with Pipelined XNOR
• Python Script Automation
• Test Results
• Future Outlook
Why One-hot?

- One-hot is fast (one-bit state look-up)
- XNOR can detect multiple-bit errors (most of the time)
- XNOR is moderate in area consumption
- XNOR tree lends itself to pipelining
- But there are trade-offs…
Pipelining Trade-off

- Reset Delays
- Logic levels
- Deterministic Level
- Chance of SBE
- $L_{XNOR}$: XNOR w/o pipeline
- $L_{FSM}$: FSM w/o XNOR
Overview

- Background Information
- Asynchronous SEU Analysis
- Existing Mitigation Methods
- Challenge
- One-Hot with Pipelined XNOR
- Python Script Automation
- Test Results
- Future Outlook
Ease of Coding

- Pipelined XNOR require a lot of hand coding in HDL
- More coding can lead to more bugs
- A code-generation tool written in Python to help adapt to our approach
- Designer describe the state transitions in a tabular form
- A Python script parses the state-transition table, and generates HDL with binary, h-3, h-2, or one-hot.
- Synthesis directives are added automatically to prevent logic pruning and FSM optimization.
Sample State-Transition Table

<table>
<thead>
<tr>
<th>#From</th>
<th>To</th>
<th>Condition</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>st1</td>
<td>.</td>
<td>Reset state</td>
</tr>
<tr>
<td>st1</td>
<td>st2</td>
<td>start</td>
<td></td>
</tr>
<tr>
<td>st2</td>
<td>st3</td>
<td>cond1</td>
<td>cond1 has higher priority than cond2</td>
</tr>
<tr>
<td>st2</td>
<td>st4</td>
<td>cond2</td>
<td></td>
</tr>
<tr>
<td>st3</td>
<td>st5</td>
<td>cond3</td>
<td></td>
</tr>
<tr>
<td>stZ</td>
<td>st1</td>
<td>condZ</td>
<td></td>
</tr>
</tbody>
</table>
What about Outputs?

• The state-transition table does not express how outputs are generated, nor do the python scripts (future work)

• Outputs can be added by:
  – Directly modifying the generated code
  – Export the current state as an output port, add output logics in higher level module
Sample Generated Code

```verilog
// Warning: State st6 is not reachable!
// Warning: State st11 is not reachable!
// Warning: State st12 is not reachable!
// Warning: State st14 is not reachable!
// Warning: State st15 is not reachable!

/*
 * fsm generated by fsm_gen
 * useEncoding   = onehot
 * useSafe       = True
 * useSafePipeline = 1
 */
	
`timescale 1ns/100ps
module fsm_sample (/*AUTOARG*/
	// Outputs
	state, nextState,
	// Inputs
	rst_an, rst, clk, ctrl0, ctrl1, ctrl2, ctrl3, ctrl4, ctrl5, ctrl6, ctrl7
);

input rst_an;
input rst;
input clk;

input ctrl0;
...
output [15:0] state;
output [15:0] nextState;

parameter st0   = 4'ho;
...
parameter st14  = 4'he;
parameter st15  = 4'hf;

reg [15:0] state /* synthesis syn_preserve=1 */;
reg [15:0] nextState;

always @ (posedge clk or negedge rst_an)
if (~rst_an) state <= 1;
else if (rst) state <= 1;else state <= nextState;

wire badState;// next-state logic (combinational)
always @ (/*AUTOSENSE*/badState or ctrl0 or ctrl1 or ctrl2
or ctrl3 or ctrl4 or ctrl5 or ctrl6 or ctrl7 or state) begin
nextState = 0;
if (badState) nextState[st0] = 1'b1;
else  case (1'b1)
// synthesis full_case parallel_case
state[st0]:
if (ctrl4) nextState[st8] = 1;
else if (ctrl1) nextState[st13] = 1;
else nextState[st0] = 1;
state[st1]:
if (ctrl0) nextState[st2] = 1;
else if (ctrl1) nextState[st3] = 1;
else if (ctrl1) nextState[st4] = 1;
else if (ctrl7) nextState[st10] = 1;
else nextState[st1] = 1;
...
...
state[st15]:
if (ctrl0) nextState[st1] = 1;
else if (ctrl0) nextState[st7] = 1;
else if (ctrl4) nextState[st10] = 1;
else nextState[st15] = 1;
endcase
end
```
// Safe logic to make sure the state register is always one-hot.
// If invalid states are detected, badState is asserted
wire [15:0] stage0 = state[15:0];
wire [7:0] stage1 = { func_add(stage0[3:0]),
    func_add(stage0[7:4]),
    func_add(stage0[11:8]),
    func_add(stage0[15:12]) } /* synthesis syn_keep=1 */;
wire [3:0] stage2 = { func_add(stage1[3:0]),
    func_add(stage1[7:4]) } /* synthesis syn_keep=1 */;
reg [0:0] stage3;
wire pre_stage3 = { func_1hot(stage2[3:0]) } /* synthesis syn_keep=1 */;
always @ (posedge clk or negedge rst_an)
if (~rst_an)
    stage3[0:0] <= 1'b0;
else
    stage3[0:0] <= pre_stage3[0:0];
assign badState = stage3;

// A 4-2 LUT that maps to three outputs:
// 00 - there is no 1
// 01 - there is exactly one 1
// 11 - there are multiple 1's
function [1:0] func_add;
    input [3:0] din;
    case (din[3:0])
        4'b0000: func_add = 2'b00;
        4'b0001: func_add = 2'b01;
        4'b0010: func_add = 2'b01;
        4'b0100: func_add = 2'b01;
        4'b1000: func_add = 2'b01;
    default: func_add = 2'b11;
    endcase // case(din[3:0])
endfunction

// The last stage function: 4-input XNOR
function func_1hot;
    input [3:0] din;
    case (din[3:0])
        4'b0001: func_1hot = 1'b0;
        4'b0010: func_1hot = 1'b0;
        4'b0100: func_1hot = 1'b0;
        4'b1000: func_1hot = 1'b0;
    default: func_1hot = 1'b1;
    endcase // case(din[3:0])
endfunction
Overview

• Background Information
• Asynchronous SEU Analysis
• Existing Mitigation Methods
• Challenge
• One-Hot with Pipelined XNOR
• Python Script Automation
• Test Results
• Future Outlook
Performance Evaluation - Setup

- 50 16-state and 50 32-state synthetic FSM are generated by a random FSM generator in state-transition tabular form
- Use the Python script to generate HDL code for the following:
  - Binary FSM
  - Hamming-3 FSM
  - Hamming-2 FSM
  - One-hot FSM
  - One-hot FSM with XNOR
  - One-hot FSM with XNOR, 1-stage pipeline
  - One-hot FSM with XNOR, 2-stage pipeline
- Synthesize, translate, map, and place-n-route with same timing constraints on Xilinx Virtex-II
- No FSM optimization or register duplication
- Compare: average logic levels and area consumption
Test Results – Speed

Logic Levels vs. FSM Type (16 States)

<table>
<thead>
<tr>
<th>Logic Levels</th>
<th>bin</th>
<th>h3</th>
<th>h2</th>
<th>1hot</th>
<th>1hot_s</th>
<th>1hot_sp1</th>
<th>1hot_sp2</th>
</tr>
</thead>
<tbody>
<tr>
<td>h3</td>
<td>6.34</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>h2</td>
<td>3.92</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1hot</td>
<td>2.72</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1hot_s</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1hot_sp1</td>
<td>3.04</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1hot_sp2</td>
<td>2.74</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Test Results – Speed

Logic Levels vs. FSM Type (32 States)

Logic Levels

<table>
<thead>
<tr>
<th>FSM Type</th>
<th>Logic Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin</td>
<td>4.54</td>
</tr>
<tr>
<td>h3</td>
<td>8.4</td>
</tr>
<tr>
<td>h2</td>
<td>4.78</td>
</tr>
<tr>
<td>1hot</td>
<td>2.86</td>
</tr>
<tr>
<td>1hot_s</td>
<td>5</td>
</tr>
<tr>
<td>1hot_sp1</td>
<td>4</td>
</tr>
<tr>
<td>1hot_sp2</td>
<td>2.84</td>
</tr>
</tbody>
</table>
Test Results – Area

Slice Counts vs. FSM Type (16 States)

<table>
<thead>
<tr>
<th>FSM Type</th>
<th>Slice Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin</td>
<td>31.42</td>
</tr>
<tr>
<td>h3</td>
<td>101.06</td>
</tr>
<tr>
<td>h2</td>
<td>37.78</td>
</tr>
<tr>
<td>1hot</td>
<td>34.4</td>
</tr>
<tr>
<td>1hot_s</td>
<td>45.14</td>
</tr>
<tr>
<td>1hot_sp1</td>
<td>41.72</td>
</tr>
<tr>
<td>1hot_sp2</td>
<td>44.66</td>
</tr>
</tbody>
</table>
Test Results – Area

Slice Counts vs. FSM Type (32 States)

<table>
<thead>
<tr>
<th>FSM Type</th>
<th>Slice Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin</td>
<td>72.36</td>
</tr>
<tr>
<td>h3</td>
<td>372.7</td>
</tr>
<tr>
<td>h2</td>
<td>85.72</td>
</tr>
<tr>
<td>1hot</td>
<td>82.38</td>
</tr>
<tr>
<td>1hot_s</td>
<td>97.46</td>
</tr>
<tr>
<td>1hot_sp1</td>
<td>88.26</td>
</tr>
<tr>
<td>1hot_sp2</td>
<td>100.04</td>
</tr>
</tbody>
</table>
Test Result - Conclusion

- One-hot design is the fastest, though not the smallest
- XNOR produces a sizable speed overhead
- Pipelining XNOR significantly reduces the speed overhead
- XNOR produces small area overhead
- Main objectives are met
Overview

• Background Information
• Asynchronous SEU Analysis
• Existing Mitigation Methods
• Challenge
• One-Hot with Pipelined XNOR
• Python Script Automation
• Test Results
• Future Outlook
Future Outlook

• Make the python script more modular
• Code-Generation templates
• Make better guesses of the pipeline insertion point
Summary

• Asynchronous SEU can cause FSM failure even with error correction logic
• SEU faults not avoidable, but can reduce the risk
• Fast logic = more reliable
• Slow logic = more prone to asynchronous SEU error, less deterministic
• XNOR can be tricky to implement
• Pipelining improves XNOR speed with trade-off

Faster is Safer!
Questions and Answers
Acknowledgements

The work described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.
Bonus Slide: Multiple-Hot

• To improve scalability of one-hot, multiple-hot encoding can be used.
• Multiple-hot is a compromise between speed and scalability.
• Can use the same error-checking mechanism as one-hot.
Bonus Slide: Multiple-Hot Examples

- 16-bit encoding, partitioned to 4 slices
  - Each slice is a 4-bit one-hot

- 16-bit encoding partitioned to 2 slices
  - Each slice is 8-bit two-hot

- 0100 1000 0001 0010
  - Up to 256 States!

- 0001 1100 0000 0110
  - Up to 784 States!
Dense vs. Sparse Encoding

Dense encoding:
- **Advantage:**
  - Compact (less flip-flops)
  - Lower chance of Single-Bit Error (SBE)
- **Disadvantage:**
  - When bit-flips happen, higher chance that FSM go to a legal state

Sparse encoding:
- **Advantage:**
  - Less likely to go to another legal state when bit-flips occur
- **Disadvantages:**
  - Consume more flip-flops
  - Higher chance of SBE