The Details of Triple Modular Redundancy
An Automated Mitigation Methodology for I/O Signals

Roger Do
FPGA Solutions Marketing
Design Creation & Synthesis Division
Methods for TMR

- Manual Coding Methods
- Automated Post-Synthesis TMR Insertion
- Automated In-Synthesis TMR Insertion and Optimization
Manual Mitigation Coding

- Difficult to code correctly
- Error-prone
- Not easily verifiable
- High Quality of Results penalty

(Manually DTMR, 60 lines of code)

```
module sample_dtmr (clk, rst, counten, countup, carryout);
  input clk, rst, counten, countup;
  output carryout;
  `ifdef manual_dtmr
    integer i;
    reg carryout_tmr0, carryout_tmr1, carryout_tmr2;
    // pragma attribute carryout_tmr0 preserve_driver true
    // pragma attribute carryout_tmr1 preserve_driver true
    // pragma attribute carryout_tmr2 preserve_driver true
    reg [7:0] count_tmr0, count_tmr1, count_tmr2;
    // pragma attribute count_tmr0 preserve_driver true
    // pragma attribute count_tmr1 preserve_driver true
    // pragma attribute count_tmr2 preserve_driver true
    reg [7:0] count_voter_tmr0, count_voter_tmr1, count_voter_tmr2;
    // pragma attribute count_voter_tmr0 preserve_driver true
    // pragma attribute count_voter_tmr1 preserve_driver true
    // pragma attribute count_voter_tmr2 preserve_driver true
    always @ (count_tmr0 or count_tmr1 or count_tmr2)
      for (i = 0; i <= 7; i = i + 1) begin
        count_voter_tmr0[i] <= ((count_voter_tmr0[i] & count_voter_tmr1[i]) &
          (count_voter_tmr0[i] & count_voter_tmr2[i]));
      end
  endmodule
```

```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
Post-Synthesis TMR

- Quality of Results impact can be “too late” after synthesis optimizations are made
- Synthesis needs to be directed to avoid certain inferencing
- Requires high level of expertise
  — Tool and techniques
Automated TMR Schemes

- **Local TMR**
  - Protects against Single Event Upsets (SEU)

- **Distributed TMR**
  - Protects against SEU & Single Event Transients (SET)

- **Global TMR**
  - Protects against SEU, SET, & Globals
  - Configuration Memory Upsets masked by DTMR or GTMR

- **Modular control allows flexibility for protection and performance**
Advanced Voter Insertion

- TMR is more than just triplication

- Voter insertion only for fault retention paths

- Voter insertion control
  - Eliminate voters to reduce delay
  - Add voters to increase protection
Smart Inferencing

- Mitigation-aware Synthesis
  - Selective DSP configuration
  - Prevent Distributed RAM inferencing
  - Prevent SRL mapping

- Better structures for design protection

- Control for design functionality and performance

Multiply-Accumulate – MAC

\[ c \leq a \times b + c; \]

Will not infer MAC... will infer multiple-add

Multiply-Accumulate – MAC
I/O Strategies

- I/O Considerations
  - Preventing single points of failure
  - Meeting I/O timing requirements
  - Output voting
Output Voter Insertion

- Performance vs. Protection
- Voter insertion after output register prevents IOB register mapping
  — Off chip timing compromised
- Voter insertion before output register allows IOB register mapping
  — IOB register is now single point of failure
  — I/O cells could have different failure rate than logic cells
- Control voter insertion to meet performance or protection requirements
I/O Triplication

- Reduces the single point of failure
- Input and Output ports/pins can be tripled
- Voter insertion control available to enable IOB register mapping
Triplicated Output Minority Voting

- Improve performance on output path
- Minority voting allows output register mapping into IOB
- Voting is not done on data path
  - Voter controls tri-state output enable
  - SEE will cause output to Hi-Z
  - Other 2 signals will pull output signal to correct value on PCB trace
Automated TMR Methods

- Full control on TMR insertion methods
  - Control for Protection vs. Performance
  - Multiple strategies for meeting I/O requirements
- Reduce risk of functional error or improper mitigation
- Less mitigation expertise needed
  - Tool, techniques, & coding
- Minimize Quality of Results impact
- Mitigation-aware Synthesis: selective inferencing
- Multi-vendor support