Enhancing Fault Tolerance in TMR Soft RISC-V Linux SoCs through Single-point Failure Analysis PhD Defense

Andrew E Wilson

Department of Electrical and Computer Engineering

Brigham Young University, Provo, UT

This work has been supported by the I/UCRC program of the NSF under grant number 1738550, the Utah NASA Space Grant Consortium (UNSGC) and Los Alamos Neutron Science Center (LANSCE) under proposal numbers NS-2017-7574-F, NS-2017-7574-A and NS-2018-7895-A

FPGAs and Soft Processors in Space

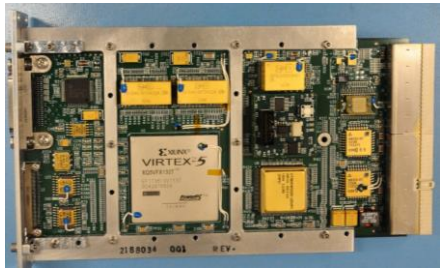
FPGAs:

- **Custom Sensor and Actuator Integration**
  Serve as reconfigurable interfaces for mission-specific sensors and actuators.
- **Onboard Processing for Performance Optimization**
  Enable low-latency, power-efficient, and high-throughput computation tailored to mission needs.
- **Support for On-Orbit Reconfiguration**
  Allow hardware logic updates post-deployment for mission adaptability and fault recovery.

Soft Processors:

- **Tightly-Coupled Control Systems**
  Enable precise coordination with adjacent hardware components and real-time interfaces.
- **Software Ecosystem Compatibility**
  Support standard operating systems, libraries, and file systems to accelerate development and reuse.
- **Safe On-Orbit Software Updates**
  Allow fault-tolerant update strategies (e.g., ping-pong schemes) with minimal mission risk.

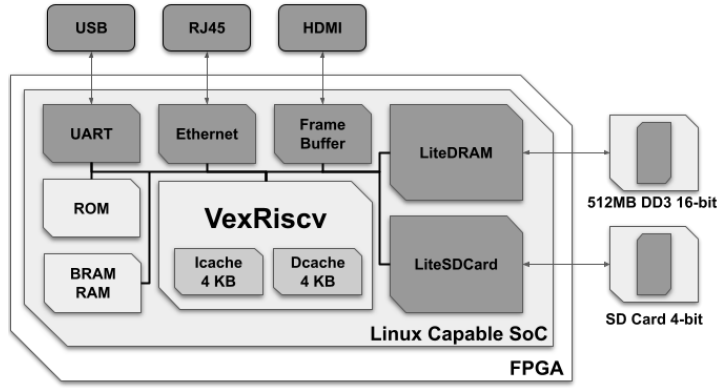
Perseverance
Mars Lander



SpaceCube 2.0

Ingenuity
Mars Helicopter

Open Standard ISA

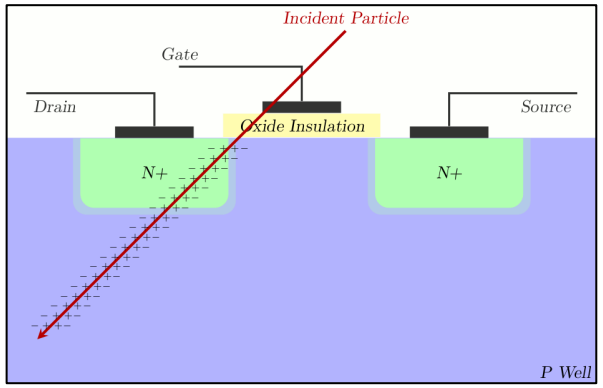


Open-Source Implementations

Radiation Effect on SRAM-based FPGAs

- **Single Event Effects (SEEs)**
  Energetic particles (e.g., protons, heavy ions) can strike sensitive regions in a transistor, causing transient or permanent effects.
- **Single Event Upset (SEU)**
  If the collected charge is sufficient to change the state of a logic element (e.g., SRAM cell), it results in a soft error.

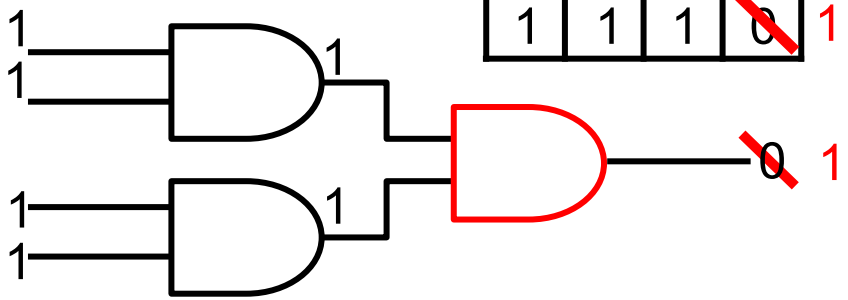
SEU



- **SEUs** flip bits in the FPGA configuration memory, potentially altering the digital design and system functionality:
  - **LUTs** – Changes logic functionality.
  - **Routing** – Connect, disconnect or short wires.
  - **Memory** – Corrupt the memory state (such as in a flip-flop or BRAM).

FPGA CRAM

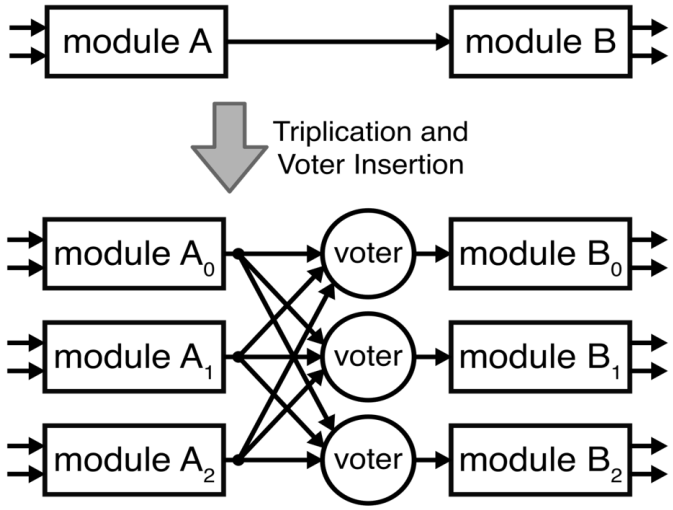
0	0	0	1
0	0	0	1
0	0	0	1
1	1	1	0 → 1



Fault Tolerance for Soft RISC-V Systems

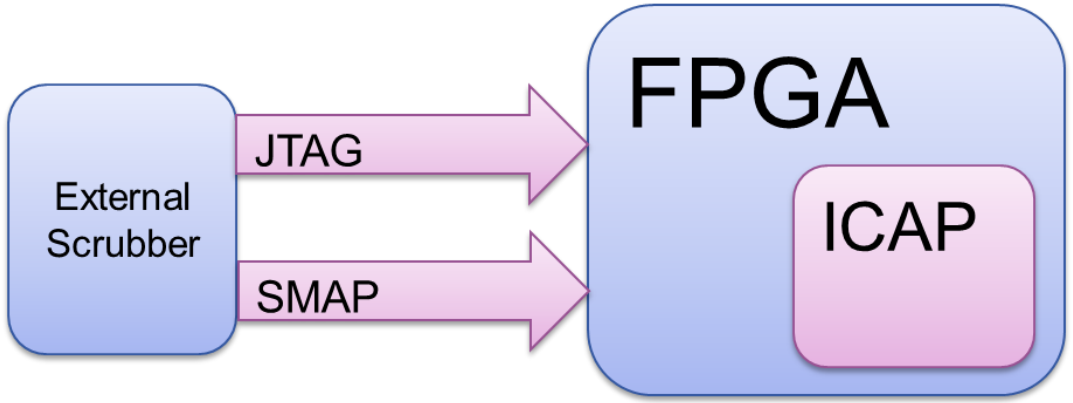
Passive Fault Tolerance with Redundancy

- **Mask:** Continue operation by hiding faults using redundancy or correction logic.
- **Distributed Triple Modular Redundancy (dTMR)**
  Replicates all logic three times and inserts triplicated majority voters between modules to ensure correct outputs when one copy fails.



Active Fault Tolerance during Operation

- **Repair:** Correct faults in hardware configuration or memory.
- **Configuration Scrubbing**
  Periodically reads and corrects upsets in the FPGA's configuration memory using a known-good bitstream or parity data.
- **Flip-Flop Resynchronization**
  Recovers from transient faults in sequential logic by reloading or re-aligning state



Fault Tolerant RISC-V SoCs in FPGAs

Problem: Redundancy Alone Is Not Always Enough for High Reliable Systems

- **Redundancy Benefits**: Can mask faults without disrupting system operation. Studies show reliability improvements ranging from **3×** [Olive et al., 2020] to **27×** [Keller et al., 2017].
- **Effectiveness Varies**: Depends on FPGA architecture, application type, and implementation strategy [Bolchini et al., 2006].
- **Synchronization Challenge**: Long-term protection requires failed modules to be resynchronized with healthy ones. Without this, redundancy degrades over time [Johnson, 2010].
- **Toolchain Limitations**: Many automated tools operate only at the functional level, ignoring physical layout constraints critical for fault tolerance [Cannon et al., 2019].
- **Design Implication**: Effective mitigation demands **fault injection**, **radiation testing**, and **systematic fault analysis** to ensure real-world reliability.

Dissertation Contributions

Goal: Create Highest Reliable RISC-V Soft Processors on SRAM FPGAs

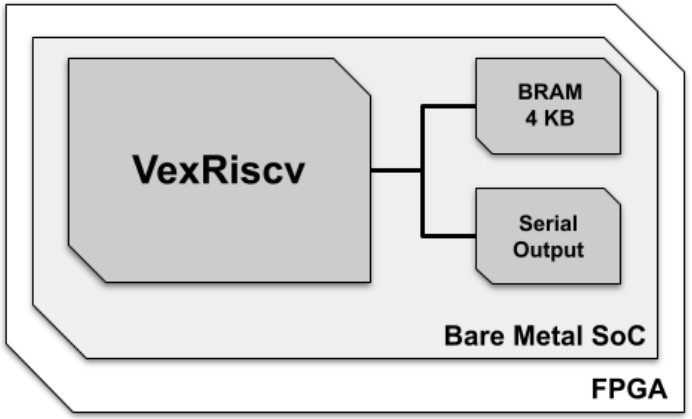
- Improved fault tolerance for RISC-V Linux SoCs on SRAM FPGAs
- Developed and applied structured fault analysis across multiple fault tolerant systems
  - Assisted by Bitstream Fault Analysis Tool (BFAT)
- Exposed TMR limitations and refined additional mitigation strategies
  - Verified 19.8× MTBF improvement for ECC implementation of open source DDR4 controller
  - Achieved 22.8× MTBF improvement with SEU-Aware placement mitigation
- Demonstrated failure analysis and identification for 98% of observed radiation-induced failures

Designs Under Test

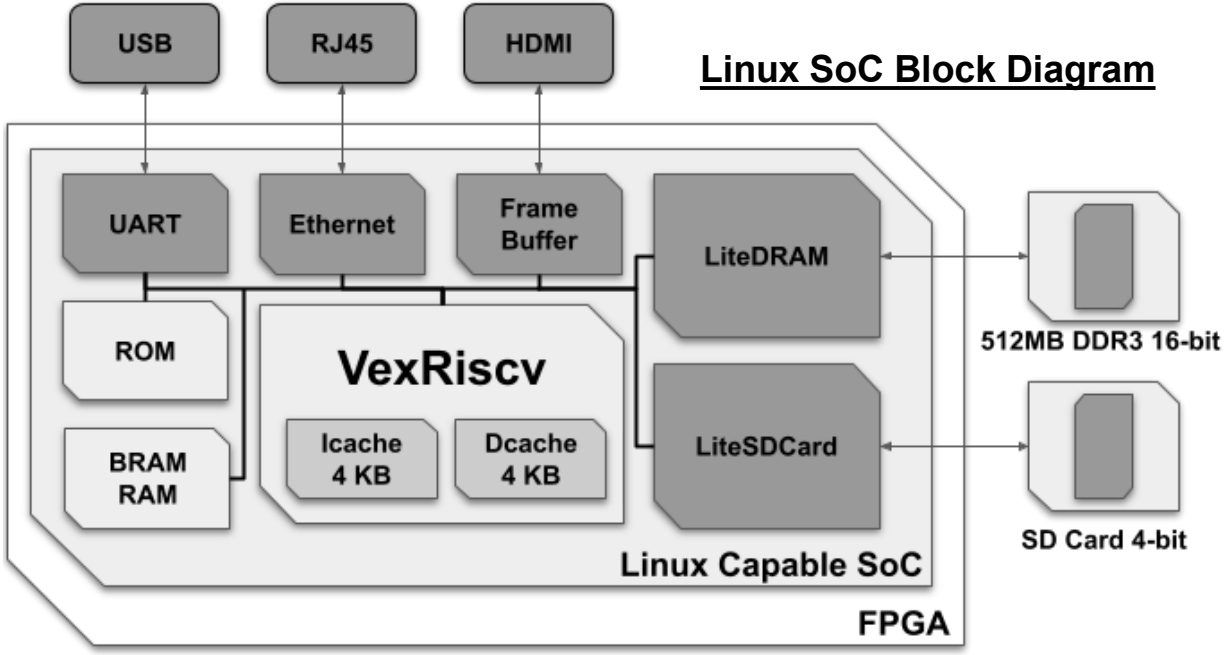
Baseline Bare Metal RISC-V Processor

- Open-source Scala VexRiscv Processor
- 4KB unified instruction and data memory
- Serial output for basic telemetry

Baseline Block Diagram



Linux SoC Block Diagram

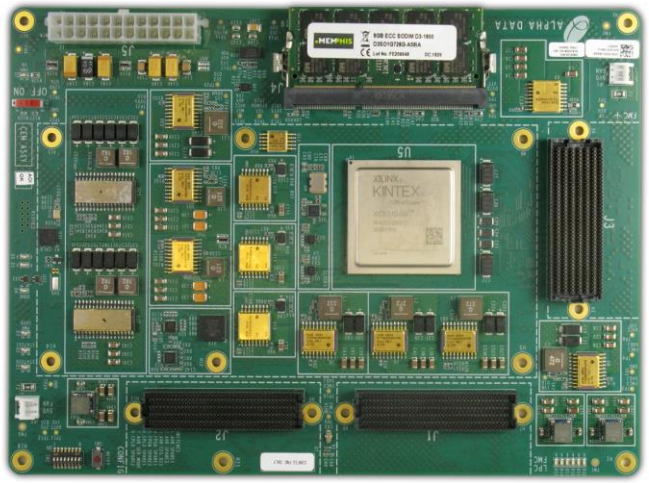


Linux RISC-V System on Chip (SoC)

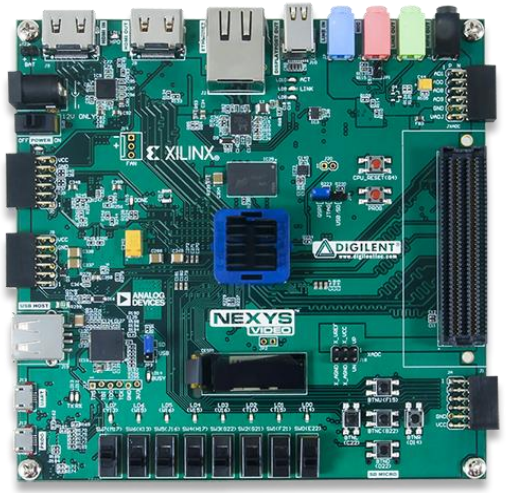
- Expanded memory with external DDR3 SRAM.
- Processor support for Instruction and Data cache with memory management to support address virtualization.
- Advanced interfaces that support different boot options.
- Buildroot embedded Linux OS kernel
- LinuxOnLitex to rapid deploy FPGA implementations

FPGA Architecture and Development Boards

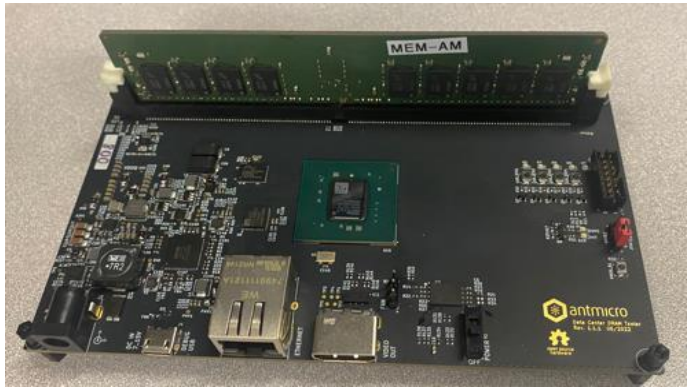
- AMD-Xilinx FPGA 7 series (28nm) and UltraScale (20nm) Architectures
- The Processor SoCs utilized the DDR3/4, SD Card, and Ethernet.
- Different development boards for different variations over the period of the work



Alpha Data SDEV Kit
AMD-Xilinx UltraScale
Bare Metal RISC-V



Nexys Video
AMD-Xilinx Series 7
Linux 16-bit DDR SoC

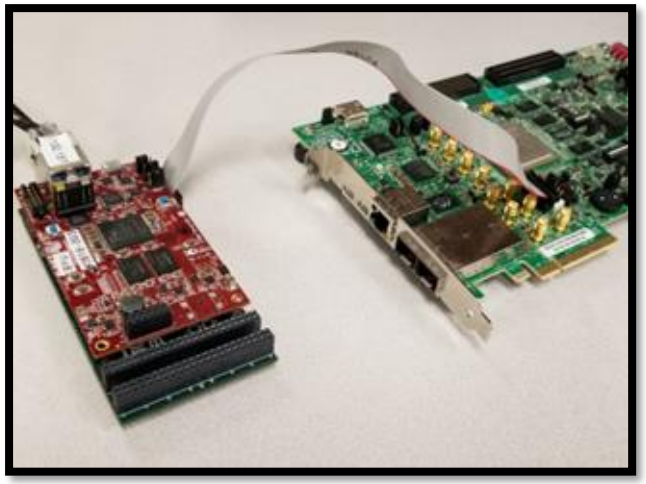


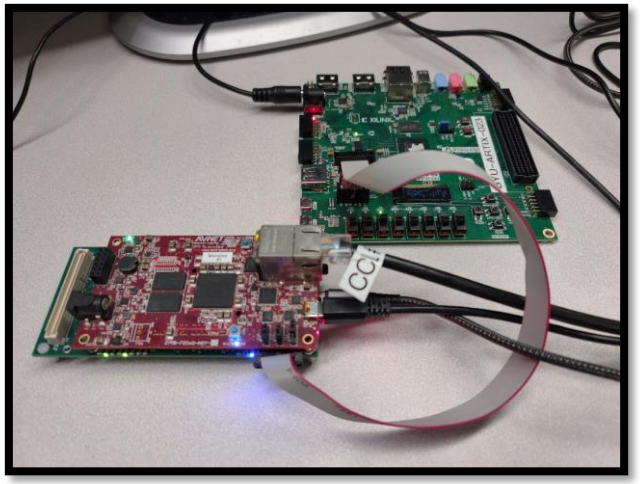
Antmicro DDR Tester
AMD-Xilinx Series 7
Linux 64-bit DDR SoC

Fault Injection for CRAM SEU Emulation

- **Purpose**: Evaluate system vulnerability by injecting controlled errors
- **Technique**: Emulate SEUs by flipping CRAM bits via JTAG commands
- **Target**: Logic & routing configuration bits (flip-flops excluded via glut mask)
- **Method**: Compare faulty vs. golden output to detect failure modes
- **Sampling**: Random fault locations provide a statistical reliability profile
- **Failure Sensitivity Formula**:

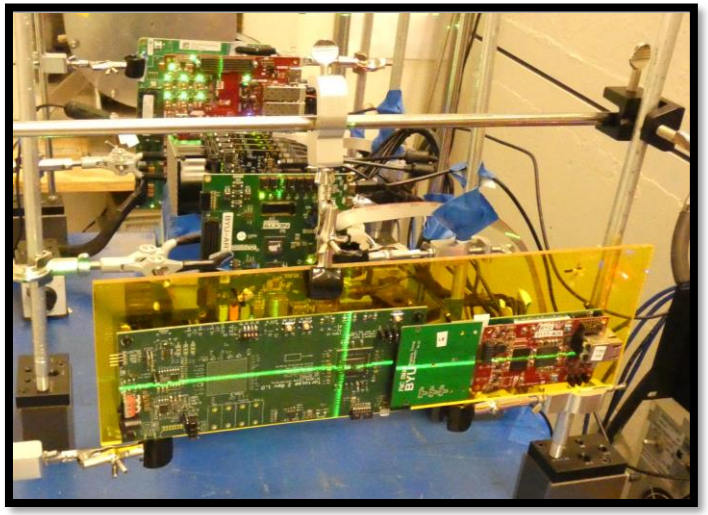
$$Failure\ Sensitivity = \frac{Number\ of\ Failures}{Number\ of\ Faults\ Injected}$$



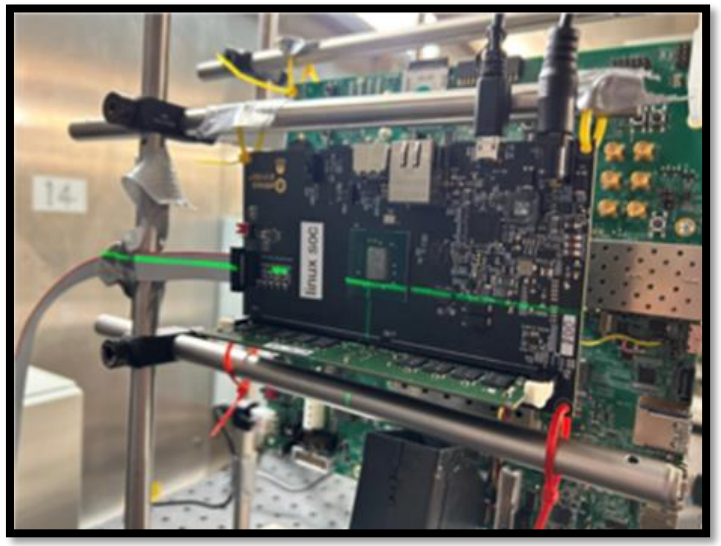


Radiation Testing

- **Purpose**: Radiation testing exposes FPGA designs to a controlled beam to simulate faults caused by space-like radiation environments.
- **Key Parameters**:
  - **Flux** (φ): Rate of radiation particles per unit area (particles/cm²/s).
  - **Fluence** (F): Total particle exposure over time, calculated as $F = \varphi \cdot t$.
- **Sensitivity Metric**: Failure sensitivity is measured via:
  - **Cross-section**: Failures per unit flux ($r = k/\varphi$).
  - **Mean Fluence to Failure**: Average exposure required to cause a fault.
- **Use Case**: Validates resilience of soft processors to real radiation-induced faults.
- **Challenges**:
  - Limited access to calibrated radiation sources and high costs restrict test scope.
  - High flux rates can overwhelm time-based or single-event-oriented mitigation techniques (e.g., scrubbing or TMR), leading to accumulated faults beyond system tolerance.



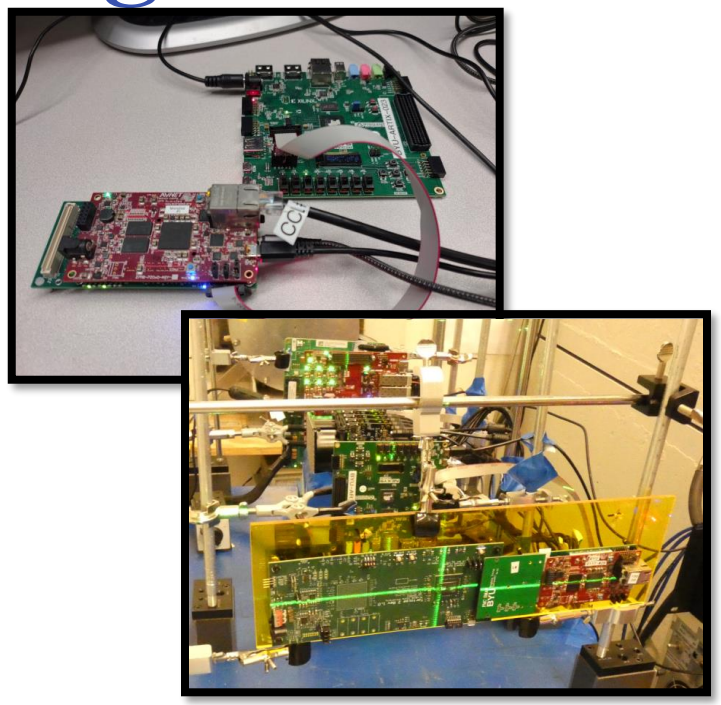
LANSCE Facility (Los Alamos, NM)



Chiplr Facility (London, UK)

Fault Injection and Radiation Testing

- **JCM Utility**: Logs CRAM upsets and performs scrubbing using a golden bitstream
- **Soft Processor Failure**: Detected via output mismatch or benchmark timeout against golden reference
- **RISC-V SoC Sensitivity**:
  - Uses **2.55× more FPGA resources,** much greater failure susceptibility
- **Root Causes of Increased Sensitivity**:
  - Higher I/O complexity
  - Larger logic footprint
  - Longer critical paths with more routing

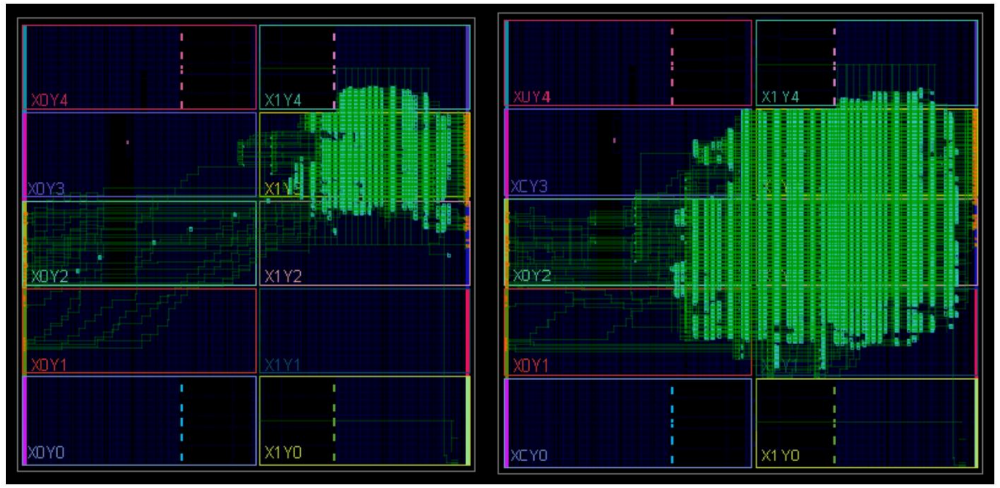


Design Under Test	Utilization (LUTs)	Design Sensitivity	Cross-Section (cm)
Bare Metal RISC-V	2,665 (1.96%)	0.05%	5.14×10^{-10}
Linux RISC-V SoC	6,791 (5.0%)	0.774%	2.09×10^{-9}
Ratio	2.55×	15.5×	4.00×

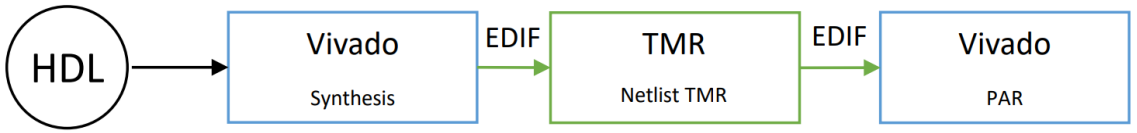
Fault Tolerance through TMR and Scrubbing

- **Tool Purpose**: SpyDrNet enables netlist parsing and transformation for FPGA designs.
- **TMR Insertion**: Supports automated distributed TMR (dTMR / fine-grain TMR) at the netlist level.
- **Flip-Flop Feedback**: Injects feedback loops into flip-flops to support next-cycle resynchronization.
- **Customizability**: Designers can target specific modules, logic types, or hierarchy levels.
- **Scalability**: TMR is applied programmatically, avoiding manual duplication and wiring.

TMR Utilization Floorplan



SpyDrNet Tool Flow



TMR Utilization Table

Design	LUT	LUTRAM	FF	BRAM
16-bit non-TMR	6,791 (5.0%)	319 (0.7%)	5,506 (2.1%)	43.0 (11.8%)
16-bit TMR	27,916 (20.9%)	957 (2.1%)	16,512 (6.2%)	129.0 (35.3%)
Cost Ratio	4.11×	3.00×	3.00×	3.00×

TMR Fault Tolerance Results

Fault Injection Results

- **Unmitigated Sensitivity**: 0.774%
  - Baseline reliability (1.00×)
- **TMR Sensitivity**: 0.055%
  - **14.07× improvement**

Design	DDR Type	Injections	Failures	Sensitivity	CV	Improvement
16-bit	Unmitigated	41,206	319	0.774%	0.056	1.00×
	TMR	504,258	280	0.055%	0.060	14.07×

Radiation Testing @ LANSCE 2021

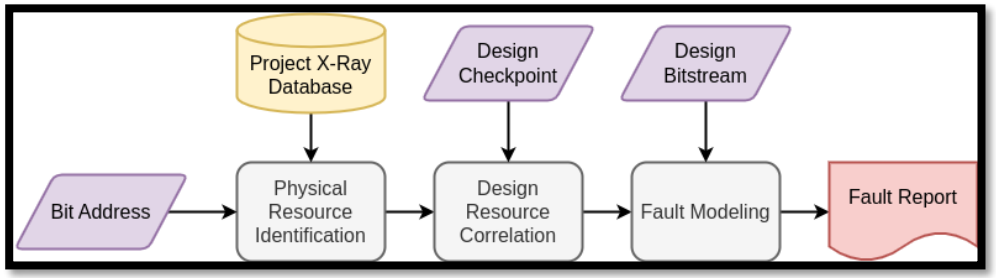
- **Unmitigated**
  - 76 failures @ 3.64×10^{10} n/cm²
  - Cross Section: **2.09×10^{-9} cm²**
- **TMR Mitigated**
  - 27 failures @ 1.91×10^{11} n/cm²
  - Cross Section: **1.42×10^{-10} cm²**
- **MTBF Gain**: **14.7×** improvement

Design	Fluence (n/cm^2)	Observed CRAM Upsets	Failures	Cross Section (cm^2)	+95% Confidence -95% Confidence	Reduction
Linux-VexRiscv Unmitigated	3.64×10^{10}	11908	76	2.09×10^{-9}	2.57×10^{-9} 1.61×10^{-9}	1×
Linux-VexRiscv TMR	1.91×10^{11}	59014	27	1.42×10^{-10}	3.47×10^{-10} 4.82×10^{-11}	14.72×

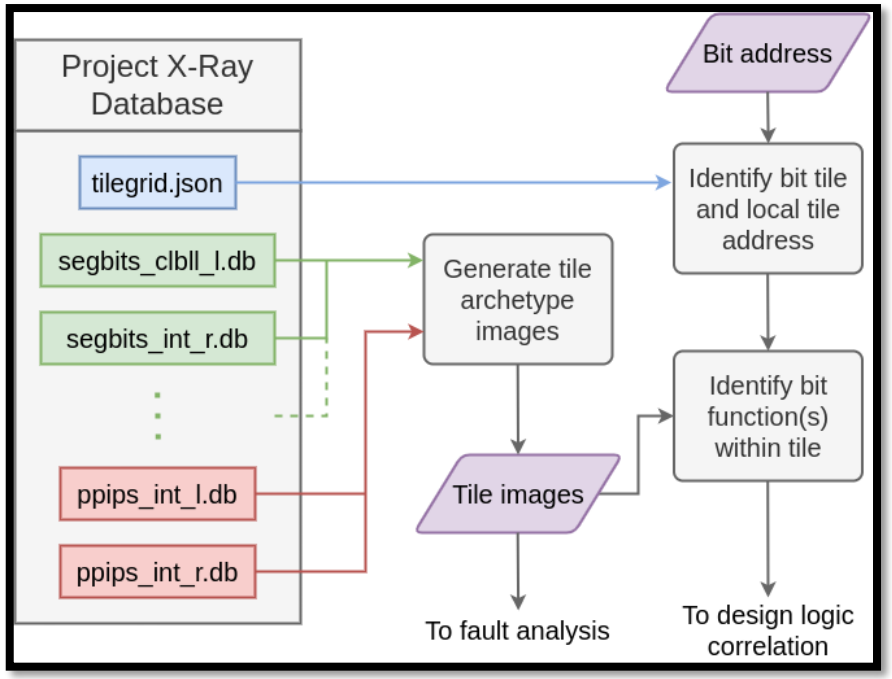
- **What is MTBF?**
  - **Mean Time Between Failures (MTBF)** estimates the average time a system operates before failing.
  - Higher MTBF means **greater reliability**—TMR allows the system to run **~14.7× longer** on average without failure.

Fault Analysis Methods

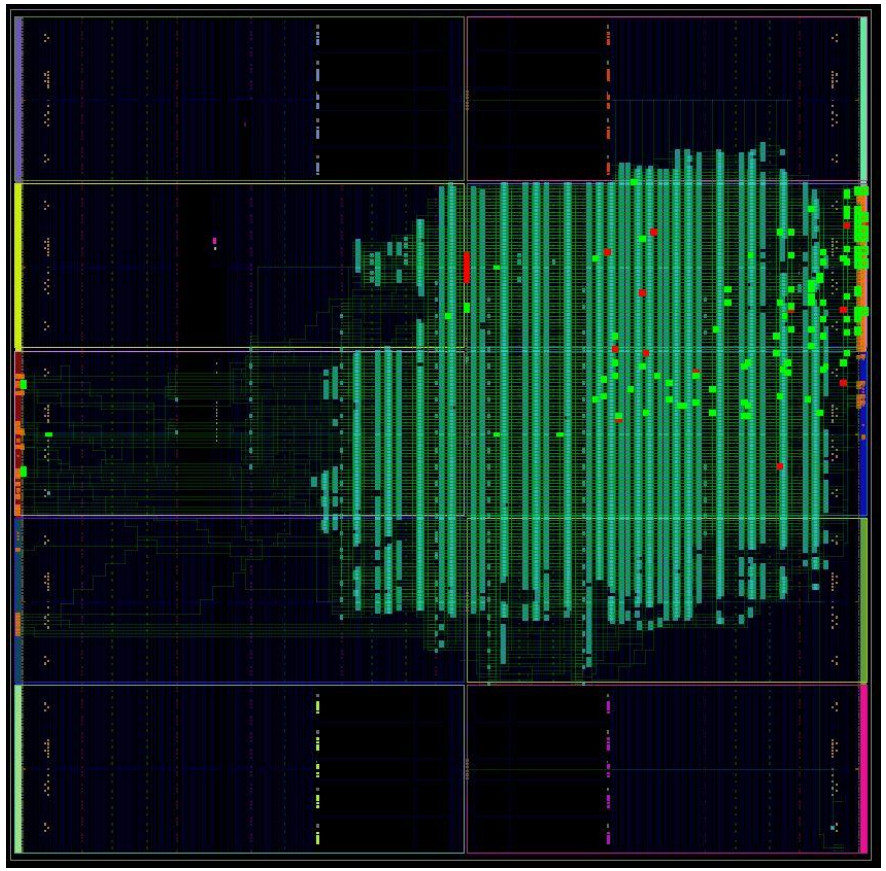
- **Tile Analysis**
  - Maps failure locations using frame address and logical tile mapping
  - Highlights critical areas (e.g., DDR I/O, clock trees, unmitigated logic)
- **BFAT (Bitstream Fault Analysis Tool)**
  A suite of Python-based static analysis tools applied to post-implementation DCPs:
  - **1. FPGA Resource Analysis**
    - Classifies failures by resource type: interconnect, I/O, clocking, CLBs
  - **2. Design Function Analysis**
    - Maps failures to logical modules (e.g., CPU, DDR controller, I/O)
  - **3. Static Net Sensitivity Analysis**
    - Identifies vulnerable, untriplicated PIPs and BELs
    - Excludes TMR, clock, and global nets



BFAT Flow



TMR Linux RISC-V 16-bit DDR BFAT Report



Tile-Based Fault & Resource Analysis (FPGA 2023)

- **Red** = radiation failures, **Green** = injection failures
- Most failures occur in **DDR I/O region** and **clock network tiles**
- **Over 70%** of failures caused by **interconnect faults**
- **Over 80%** tied to **DDR interface activity**
- TMR tools can't protect most interconnect faults related to the I/O primitives
- Next Slide: Example of a mapped fault in the implemented design related to the DDR interface

Resource	Occurrences (radiation test)	Occurrences (random fault injection)	Occurrences (total)	Estimated Bits
Interconnect	13 (76.4%)	75 (70.8%)	88 (71.5%)	17,855
I/O Pin	3 (17.6%)	20 (18.9%)	23 (18.7%)	4,667
Clocking	1 (5.9%)	6 (5.7%)	7 (5.7%)	1,420
CLB	0 (0.0%)	4 (3.8%)	4 (3.3%)	812
Undefined	0 (0.0%)	1 (0.9%)	1 (0.8%)	203
Total	17	106	123	24,956 (0.042%)*

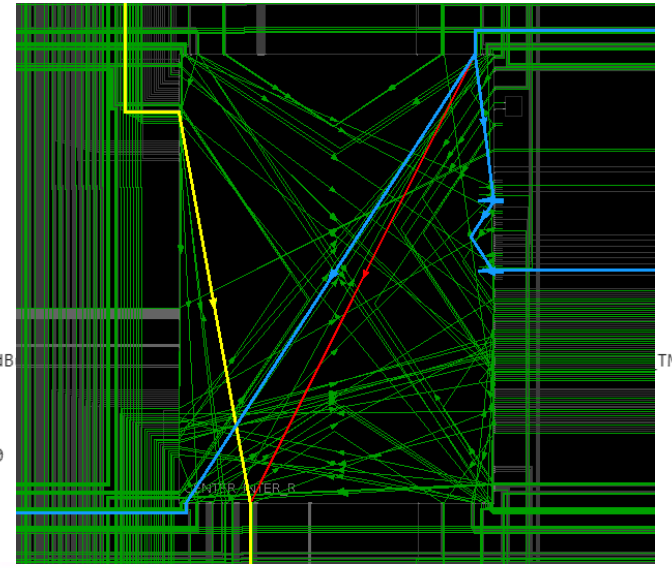
* Percentage is in comparison to the 59,145,600 type 0 CRAM locations in the XC7A200T

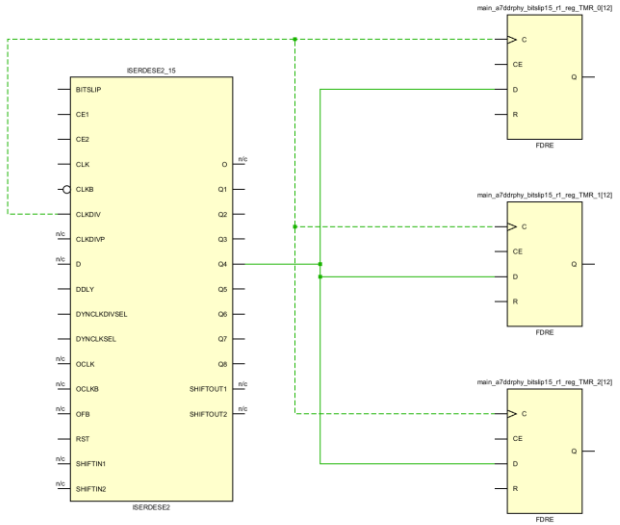
Table 6.2: FPGA Device Resources Causing Design Failure.

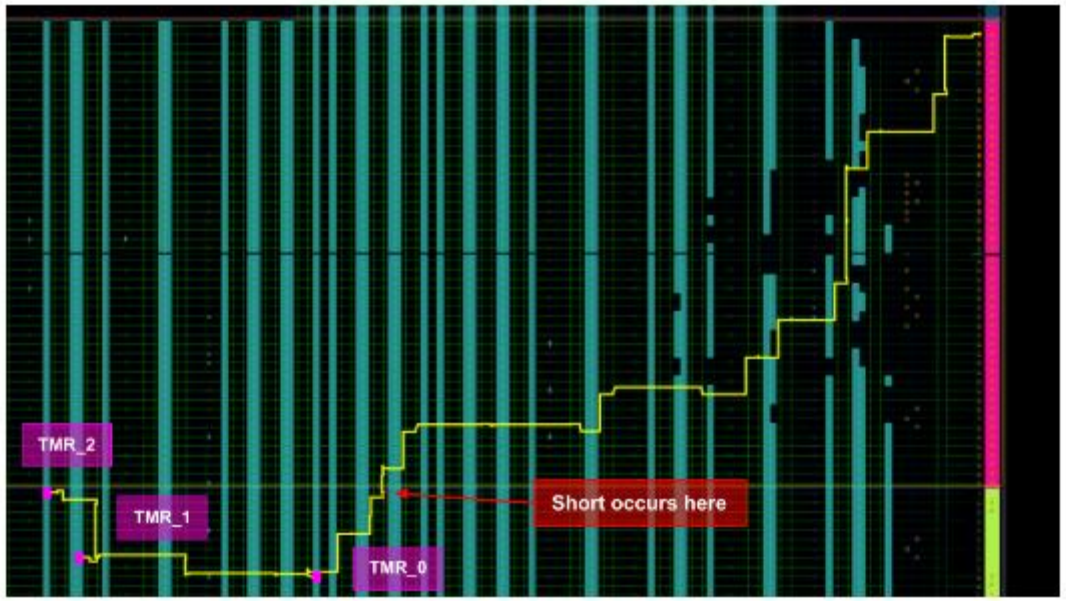
DDR Input Upset in Routing

```
================================================================================
                            Bit Group 1
================================================================================

Failure Bits:
------------------------------
bit_00402785_100_15 (0->1)
        INT_R_X79Y149 - SW6BEG2 2-20 Routing Mux - Column Bit
        Resource Design Name: INT_R_X79Y149/SW6BEG2
        Shorts formed between net(s): ISERDESE2_15_n_4 (initially connected), VexRiscvLitexSmpCluster_Cc1_Iw32Is4096Iy1_Dw32Ds4096Dy1_ITs4DTs4_Ldw128_Ood/dB                                                    TMR_0[14]
        Affected PIPs:
                WW2END2->>SW6BEG2 (activated)
        Affected Resources:
                VexRiscvLitexSmpCluster_Cc1_Iw32Is4096Iy1_Dw32Ds4096Dy1_ITs4DTs4_Ldw128_Ood/dBridge_logic/io_output_rdata_fifo/storage_12_reg_10_i_12_TMR_0
                VexRiscvLitexSmpCluster_Cc1_Iw32Is4096Iy1_Dw32Ds4096Dy1_ITs4DTs4_Ldw128_Ood/dBridge_logic/io_output_rdata_fifo/storage_12_reg_8_i_10_TMR_0
                VexRiscvLitexSmpCluster_Cc1_Iw32Is4096Iy1_Dw32Ds4096Dy1_ITs4DTs4_Ldw128_Ood/dBridge_logic/io_output_rdata_fifo/storage_12_reg_8_i_10_TMR_1
                main_a7ddrphy_bitslip15_r1_reg_TMR_0[12]
                main_a7ddrphy_bitslip15_r1_reg_TMR_1[12]
                main_a7ddrphy_bitslip15_r1_reg_TMR_1[4]
                main_a7ddrphy_bitslip15_r1_reg_TMR_2[12]
```

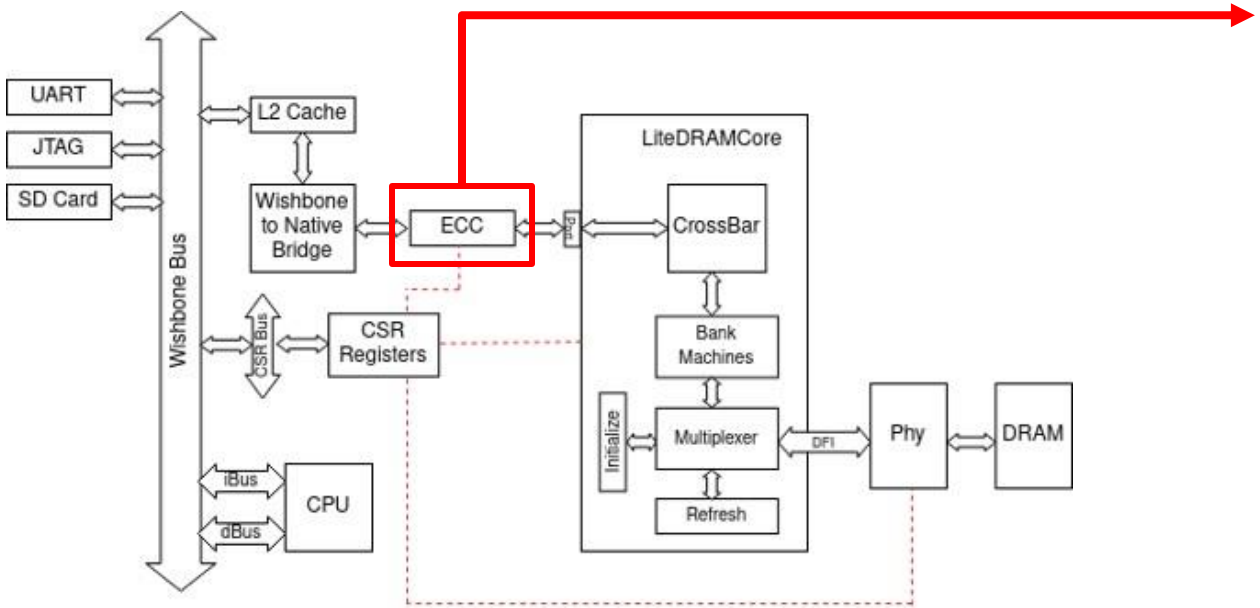


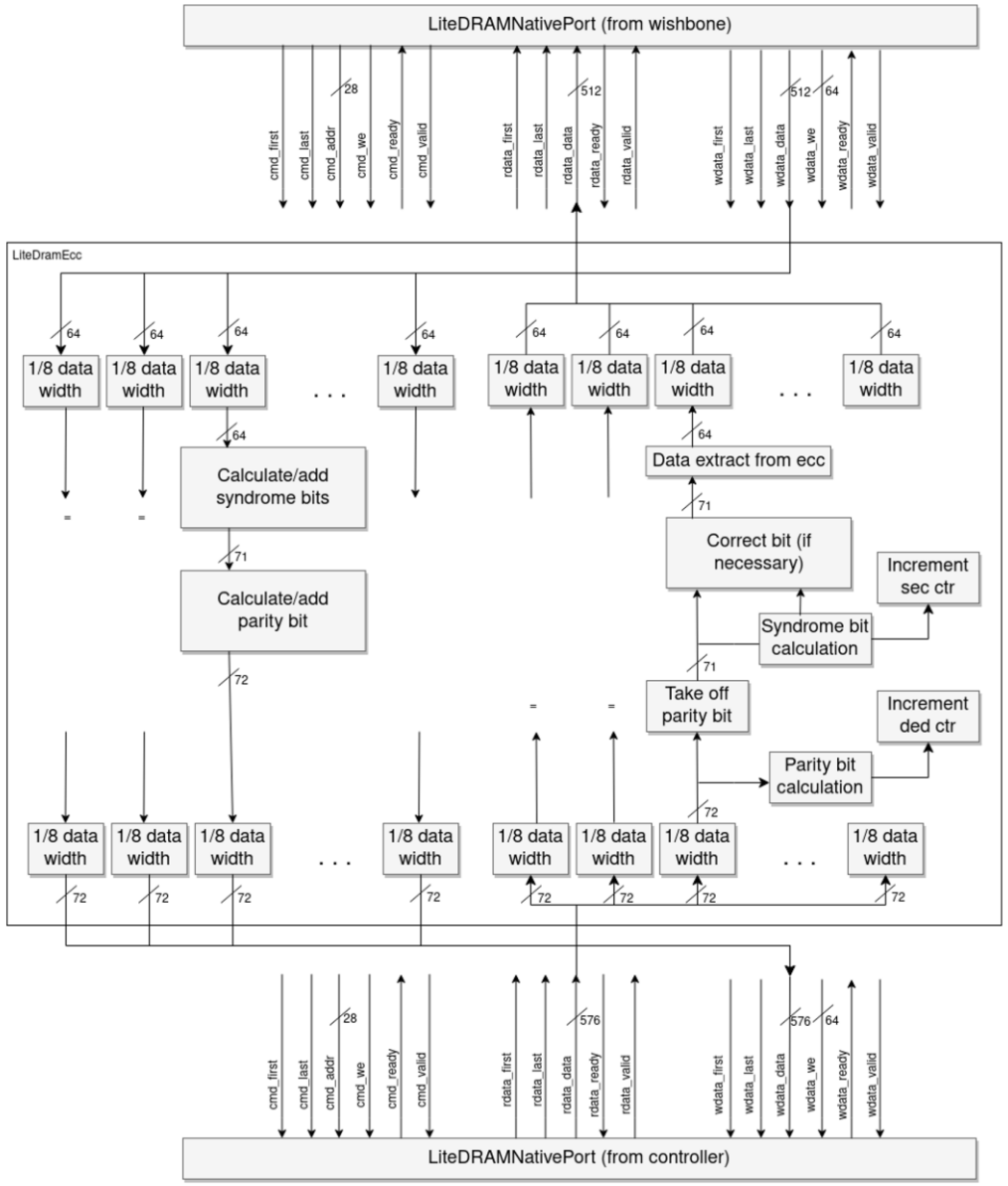




ECC for TMR 64-bit SoC

- **Enable ECC Generator**: Modify LiteDRAM core to include Hamming (or SECDED) logic for data writes.
- **Integrate ECC Checker**: Add error detection and correction on data reads before passing to SoC/memory bus.
- **Extend Data Width**: Expand memory interface to accommodate ECC bits (e.g., 64b data + 8b ECC).





TMR and ECC Fault Tolerance Results

Fault Injection Results

- TMR 64-bit DDR Sensitivity: 0.230%
  - (Baseline TMR improvement : 5.55×)
- TMR & ECC Sensitivity: 0.0035%,
  - 35.06× improvement (~7× over just TMR)

Design	Injections	Failures	Sensitivity	CV	Improvement
Unmitigated	13,183	168	1.274%	0.077	1.0×
ECC	81,299	886	1.090%	0.033	1.17×
TMR	54,835	126	0.230%	0.089	5.55×
TMR & ECC	220,687	78	0.035%	0.113	36.06×

Radiation Testing @ ChipIr 2023

- Unmitigated:
  - 161 failures @ 4.21×10^{10} n/cm²
  - Cross Section: 3.82×10^{-9} cm²
- TMR-Protected:
  - 105 failures @ 5.45×10^{11} n/cm²
  - Cross Section: 1.93×10^{-10} cm²
- MTBF Gain: 19.84×

Design	Fluence (n/cm²)	Observed CRAM Upsets	Failures	Cross Section (cm²)	+95% Confidence -95% Confidence	Reduction
64-bit DDR SoC Unmitigated	4.21×10^{10}	15656	161	3.82×10^{-9}	4.42×10^{-9} 3.22×10^{-9}	1×
72-bit DDR SoC TMR & ECC	5.45×10^{11}	198225	105	1.93×10^{-10}	2.30×10^{-10} 1.55×10^{-10}	19.84×

BFAT Failure Analysis

- Interconnect single-point failures reduced by 90%

Resource	Occurrences (TMR)	Occurrences (TMR & ECC)	Estimated Bits (TMR)	Estimated Bits (TMR & ECC)
Interconnect	95 (82.6%)	31 (48.4%)	75,630	6,941
I/O Pin	10 (8.7%)	18 (28.1%)	7,961	4,030
Clocking	0 (0.0%)	10 (15.6%)	0	2,239
CLB	9 (7.8%)	5 (7.8%)	7,164	1,120
Undefined	1 (0.9%)	0 (0.0%)	796	0
Total	115	64	91,552 (0.23%)*	14,330 (0.036%)*

*Percentage is in comparison to the 39,805,312 type 0 CRAM locations in the XC7K160T

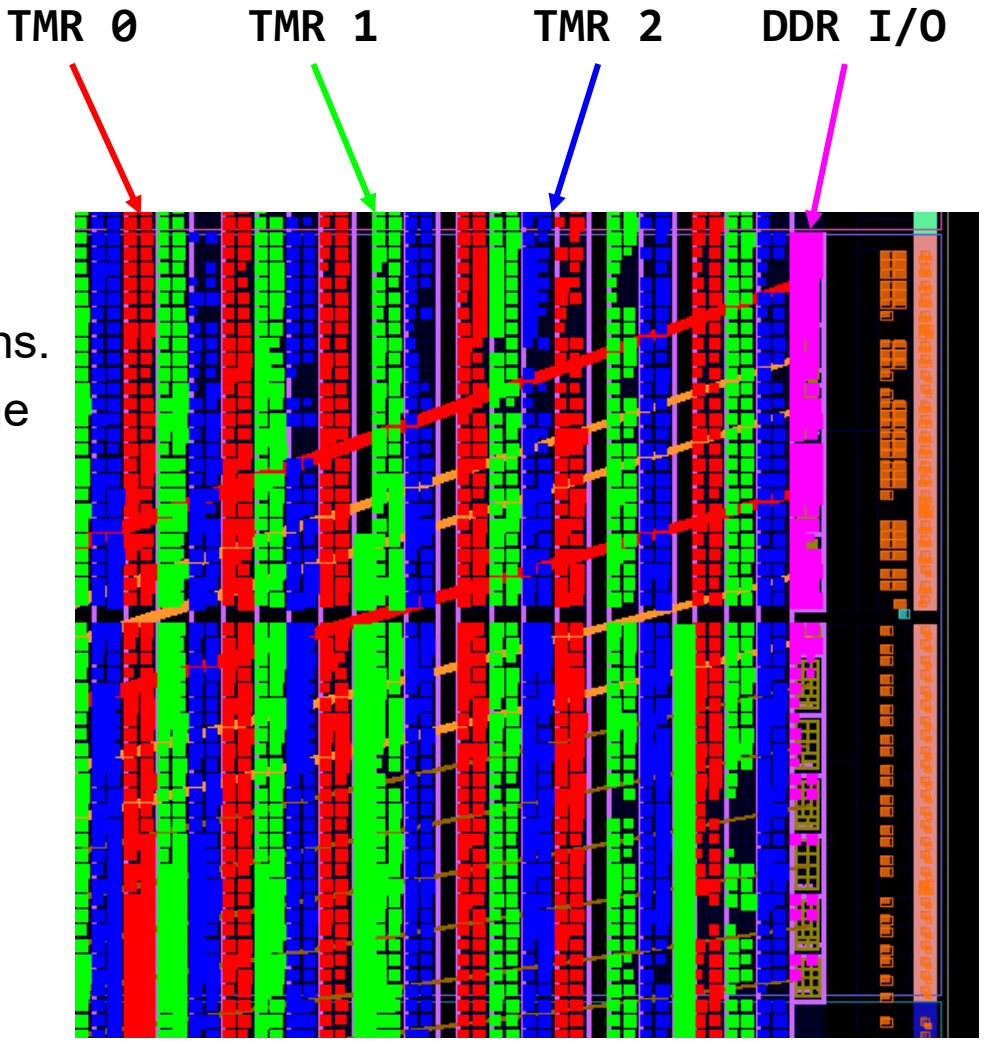
SEU-Aware Placement for TMR 16-bit SoC

Striping for TMR Isolation

- Assigns each TMR domain to a vertical FPGA stripe.
- Reduces routing overlap and PIP/switchbox sharing.
- Enforced via suffix naming (_TMR_0/1/2) + regex constraints.
- Limits the sharing of VCC/GND primitives between TMR domains.
- Improves fault isolation, but less effective in dense SoCs with the usage of special resources like BRAMs and DSPs.

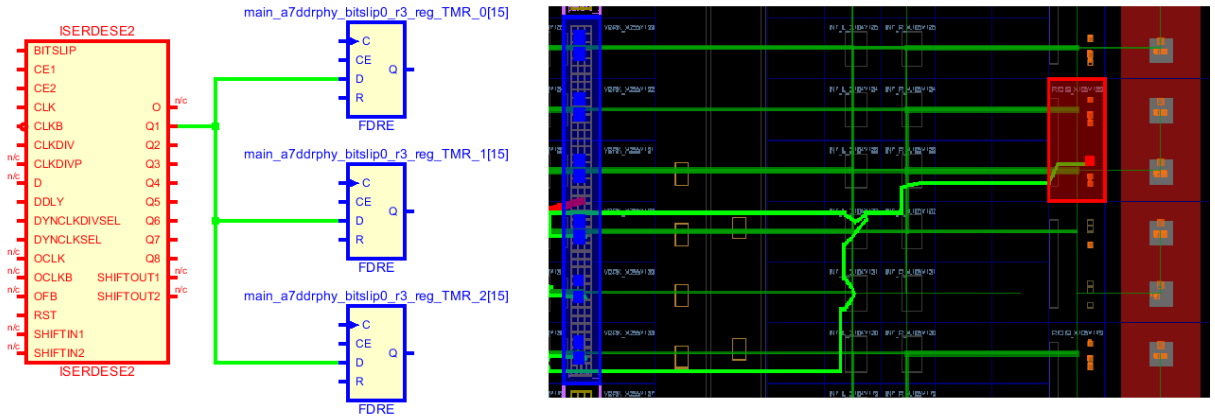
```
create_pblock pblock_dut_tmr_0
resize_pblock [get_pblocks pblock_dut_tmr_0] -add {
    SLICE_X0Y0:SLICE_X1Y249
    SLICE_X6Y0:SLICE_X7Y249
    ...
    SLICE_X96Y0:SLICE_X97Y249
    SLICE_X102Y0:SLICE_X103Y249
}
set_property IS_SOFT 0 [get_pblocks pblock_dut_tmr_0]
add_cells_to_pblock [get_pblocks pblock_dut_tmr_0] \
    [get_cells -hierarchical -regexp .*TMR(_VOTER)?_0.*
        -filter IS_PRIMITIVE==1]
```

Example Striping TCL Script



SEU-Aware Placement for TMR 16-bit SoC

Input Path Example




```
create_pblock piserd_0
add_cells_to_pblock [get_pblocks piserd_0] [get_cells -quiet [list \
    main_a7ddrphy_bitslip0_r3_reg_TMR_0[15] \
    main_a7ddrphy_bitslip0_r3_reg_TMR_1[15] \
    main_a7ddrphy_bitslip0_r3_reg_TMR_2[15]]]
resize_pblock [get_pblocks piserd_0] -add {SLICE_X162Y180:SLICE_X163Y185}
```

ISERDES TCL Example

Output Path (OSERDES + LUT Voters)

- I/O primitives (e.g., OSERDES) can't be triplicated, expose critical vulnerabilities
- **LUT voters** reduce triplicated logic to a single value before OSERDES
- Default tool placement leads to long, SEU-prone nets
- **Mitigation**: Constrain voter LUTs near OSERDES blocks to reduce PIPs
- Result: 2.5× reduction in routing sensitivity (10.4 → 4.1 PIPs)

Input Path (ISERDES + TMR Flip-Flops)

- ISERDES drives **triplicated flip-flops**, vulnerable if widely spread
- Default tool placement increases interconnect CRAM exposure
- **Mitigation**: Group flip-flops near each ISERDES block via pblocks
- Result: 3.2× reduction in PIPs (24.6 → 7.8 PIPs)

Static Analysis on Unmitigated Routes

- **Objective**: Quantify and reduce single-point failure risks in configuration memory
- **Approach**: Apply incremental mitigations and re-analyze post-implementation design
- **Tool Used**: BFAT (Bitstream Fault Analysis Tool) for net-level sensitivity tracking
- **Scope**: Focused on unmitigated PIPs, BELs, and routing CRAM bits
- **Key Result**:
  - **Unmitigated Design**: 2.95% of CRAM bits sensitive
  - **Fully Mitigated Design**: 0.056% sensitive → **37.5× reduction** in vulnerability

Design Variant	Description of Mitigations
TMR with One Clock	Baseline 16-bit TMR Linux SoC with a single shared clock tree. No placement constraints.
TMR Clocks	Triplicated BUFG primitives added, with constraints to preserve clock buffer separation.
TMR Clocks + Striping	Adds domain striping: physical separation of logic for each TMR domain via pblocks.
TMR Clocks + DDR I/O	TMR BUFGs and flip-flops around OSERDES/ISERDES blocks to reduce I/O net sensitivity.
TMR & All Mitigations	Final design incorporating all strategies: TMR clocks, domain striping, and placement-aware DDR I/O constraints.

Design	Unmitigated PIPs & BELs	Percent of CRAM*	Improvement
Unmitigated	1,238,982	2.95%	1.0×
TMR with One Clock	71,802	0.121%	17.26×
TMR Clocks	59,991	0.101%	20.65×
TMR Clocks + Striping	45,428	0.077%	27.27×
TMR Clocks + DDR I/O	31,631	0.053%	39.17×
TMR & All Mitigations	33,019	0.056%	37.52×

*Calculated by dividing the number of unmitigated PIPs and BELs by the total number of Type 0 CRAM bits in the XC7A200T device (59,145,600).

SEU-Aware PAR, TMR Fault Tolerance Results

Fault Injection Results

- TMR 16-bit DDR Sensitivity: 0.039%
  - (Baseline TMR improvement : 14.74×)
- TMR & All Mitigations Sensitivity: 0.0019%
  - 31.02× improvement

Design Variant	Injections	Failures	Sensitivity (%)	CV	Improvement
Unmitigated	24,218	139	0.574%	0.085	1.00×
TMR with One Clock	603,366	235	0.039%	0.065	14.74×
TMR Clocks	121,162	37	0.031%	0.164	18.79×
TMR Clocks + Striping	155,635	45	0.029%	0.149	19.85×
TMR Clocks + DDR I/O	103,884	25	0.024%	0.200	23.85×
TMR & All Mitigations	270,209	50	0.019%	0.141	31.02×

Radiation Testing @ LANSCE 2022

- Unmitigated:
  - 95 failures @ 3.67×10^{10} n/cm²
  - Cross Section: 2.59×10^{-9} cm²
- TMR-Protected:
  - 37 failures @ 3.26×10^{11} n/cm²
  - Cross Section: 1.14×10^{-10} cm²
- MTBF Gain: 22.78×

Design	Fluence (n/cm²)	Observed CRAM Upsets	Failures	Cross Section (cm²)	+95% Confidence -95% Confidence	Reduction
Linux-VexRiscv (Unmitigated)	3.67×10^{10}	16,239	95	2.59×10^{-9}	3.12×10^{-9} 2.06×10^{-9}	1×
Linux-VexRiscv (TMR & All Mitigations)	3.26×10^{11}	144,841	37	1.14×10^{-10}	2.70×10^{-10} 3.38×10^{-11}	22.78×

BFAT Failure Analysis

- Interconnect single-point failures reduced by 50%

Resource	Occurrences (original)	Occurrences (improved)	Estimated Bits (original)	Estimated Bits (improved)
Interconnect	88 (71.5%)	53 (57.0%)	17,855	8,762
I/O Pin	22 (17.9%)	20 (21.5%)	4,464	3,306
Clocking	7 (5.7%)	3 (3.2%)	1,420	496
CLB	4 (3.3%)	16 (17.2%)	812	2,646
Undefined	2 (1.6%)	1 (1.1%)	406	169
Total	123	93	24,956 (0.042%)*	15,379 (0.026%)*

* Percentages are in comparison to the 59,145,600 type 0 CRAM locations in the XC7A200T

Fault Analysis Fault Tolerant Evaluation

- **Intrinsic device and design constraints** cap fault tolerance at ~50× MTBF improvement
- **Residual failures** are largely due to vulnerabilities in the **DDR memory interface**
- **Unmitigatable single-point failures** include:
  - SEFIs (Single Event Functional Interrupts)
  - Critical clock routing mux failures

Failure Category	Count	Percent
Unmitigatable Failures	24.5	23.33%
XDC-Based Placement	12.0	11.43%
ADDR & CMD Parity	12.5	11.90%
DDR Error Accumulation	56.0	53.33%
Total	105	100%

Table 9.1: Failure Categories for the Last 5%

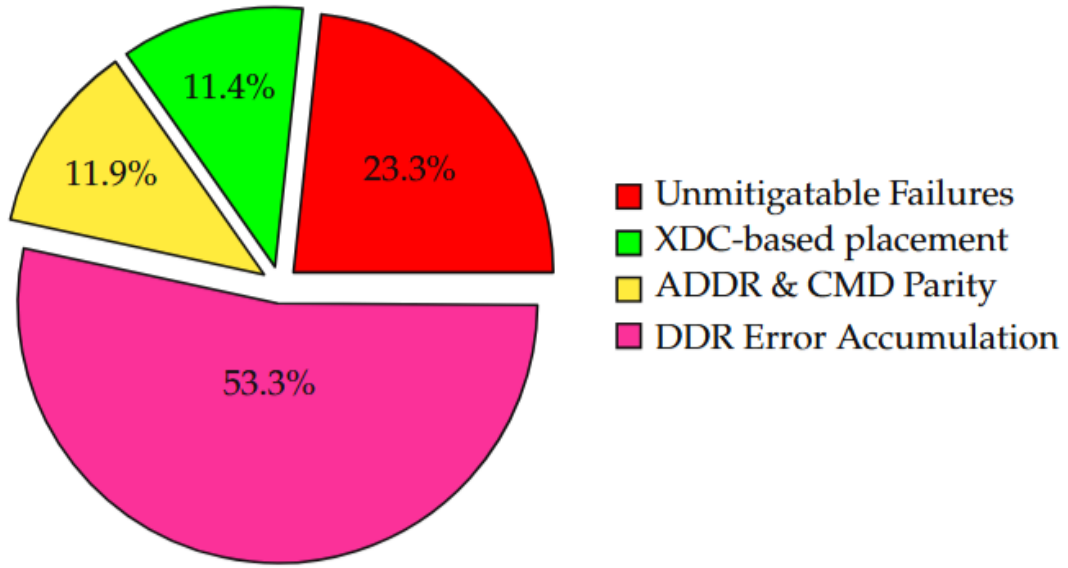


Figure 9.1: The Last 5% of Failures for the 72-bit TMR & ECC Design

Potential Mitigations

- **ECC Data Scrubbing**
  - Periodically reads and corrects memory upsets using error-correcting codes
  - A scrub-to-upset ratio of **1:10** can reduce memory-related failures by **up to 65×**
- **Distributed Reset for I/O Primitives**
  - Actively resets vulnerable I/O blocks (e.g., SERDES, IOBs) during faults
  - Prevents persistent error states without full reconfiguration
  - Especially useful for **non-triplicated I/O**, where redundancy cannot be applied

    [Cannon et al., "Improving the Reliability of TMR With Nontriplicated I/O on SRAM FPGAs," 2019]
- **DDR4 Address & Command Parity**
  - Adds parity bits to control signals (e.g., address, command) for DDR4 memory
  - Helps detect and isolate control path errors that could corrupt memory transactions

Lessons Learned

- Majority of Fault Tolerant design failures are related to FPGA physical resource usage, placement, and routing.
  - Beyond synthesis-level TMR tools capabilities.
- TMR Fault tolerance is highly dependent on design choices and should always be coupled with fault injection and analysis.
- There is still opportunity for significant fault tolerance improvement at the cost of minimal overhead and design intrusion.
  - DDR ECC scrubbing would be a **2× MTBF** incremental improvement
  - Address & Command Parity would be a **1.5× MTBF** incremental improvement
  - Distributed Reset would be a **1.5× MTBF** incremental improvement

Pushing Fault Tolerance Further

- Need SW & HDL functions beyond a simple benchmark.
  - (more realistic baseline).
- BRAM Scrubbing on ROMs and time-out on cached entries.
- Mitigation against configuration signals (i.e. xapp 1098, init_b intercept)
- Dedicated Isolated Clk sources and internal clock routing.
  - (3x MMCMs with sync, isolated routes to BUFGs, redundant DDR clock source).
- Extended control for 72-bit DDR4 with byte module-level recovery.
  - via byte module reset and RAID 5 parity scheme.
- Hardened external TMR voters for critical reset lines.
- Kernel Modification for monitoring and checkpoint recovery.
  - Focus on optimizing uptime and reducing downtime.

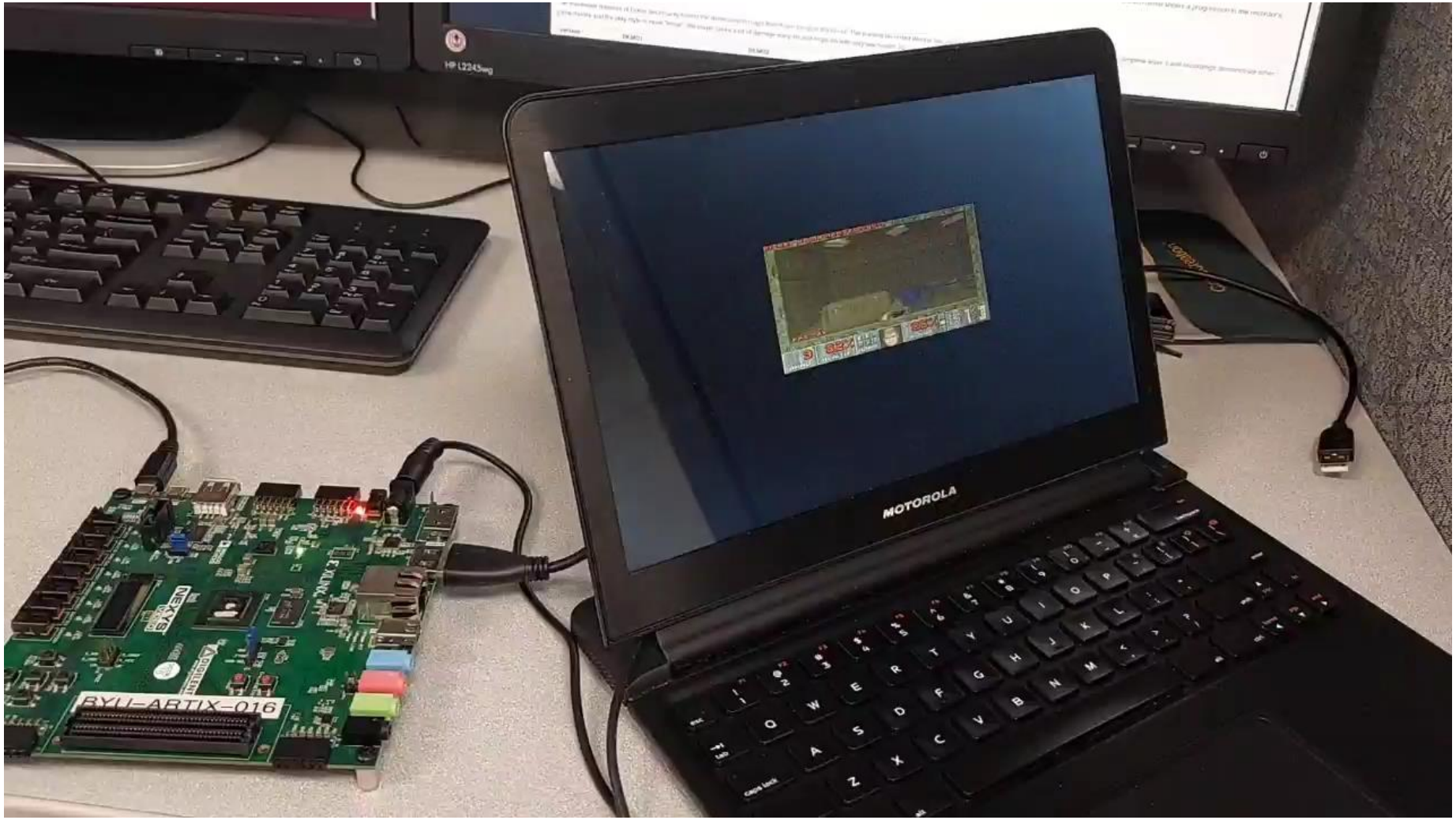
Publications and Presentations

- Primary Author
  - Neutron radiation testing of fault tolerant RISC-V soft processor on Xilinx SRAM-based FPGAs - 2019 IEEE Space Computing Conference (SCC)
  - Fault Injection Testing of Fault Tolerant RISC-V Soft Processors on Xilinx SRAM-based FPGAs – 2021 Journal of Radiation Effects Research and Engineering
  - Neutron radiation testing of a TMR VexRiscv soft processor on SRAM-based FPGAs – 2021 IEEE Transactions on Nuclear Science
  - Post-radiation Fault Analysis of a High Reliability FPGA Linux SoC – 2023 International Symposium on Field Programmable Gate Arrays
  - Neutron radiation testing of RISC-V TMR soft processors on SRAM-based FPGAs – 2023 IEEE Transactions on Nuclear Science
- Secondary Author
  - Post-Irradiation Fault Injection for Complex FPGA Designs – 2025 IEEE Transactions on Nuclear Science
- Poster
  - Neutron Radiation Results of Fault-Tolerant Soft RISC-V Linux SoCs on SRAM-based FPGAs – 2025 IEEE Nuclear and Space Radiation Effects Conference
- Conference Presentations
  - You've got to break a few FPGAs to go to Space! – 2022 Hackaday SuperCon
  - Evaluating a Cost-Effective, Open-Source, and Radiation-Tolerant FPGA SDR via Fault Injection for SmallSat Applications – 2025 SmallSat Salt Lake City

Future Work

- Reaching the Fault Tolerance limit with new mitigation methods.
  - ECC Data scrubbing.
  - Distributed reset for IO primitives.
  - Address and Command Parity for DDR4.
- Exploring newer generations of FPGA and different vendors.
- Implement automation of placement constraints with tightly coupled fault injection and analysis.
- Look at different fault tolerant methods for external memory such as Reed-Solomon error correction codes or radiation intelligent memory Controllers.

DOOM!



Thank You

- Questions Please!