

# FPGA Mitigation Strategies for Critical Applications



**Melanie Berg, SSAI in support of NASA/GSFC**  
**[Melanie.D.Berg@NASA.gov](mailto:Melanie.D.Berg@NASA.gov)**



# Acronyms

- Application specific integrated circuit (ASIC)
- Block random access memory (BRAM)
- Block Triple Modular Redundancy (BTMR)
- Clock (CLK or CLKB)
- Clock domain crossing (CDC)
- Clock period ( $t_{clk}$ )
- Combinatorial logic (CL)
- Configurable Logic Block (CLB)
- Constant false alarm rate filter (CFAR)
- Device under test (DUT)
- Digital Signal Processing Block (DSP)
- Distributed triple modular redundancy (DTMR)
- Dual interlocked storage cell (DICE)
- Edge-triggered flip-flops (DFFs)
- Error detection and correction (EDAC)
- Error rate ( $dE/dt$ )
- Field programmable gate array (FPGA)
- Finite impulse response filter (FIR)
- Gate Level Netlist (EDF, EDIF, GLN)
- Global triple modular redundancy (GTMR)
- Input – output (I/O)
- Intellectual property (IP)
- INV (inverter)
- Linear energy transfer (LET)
- Local triple modular redundancy (LTMR)
- Logic equivalency checking (LEC)
- Look up table (LUT)
- Mean fluence to failure (MFTF)
- Mean Time to Failure (MTTF)
- One time programmable (OTP)
- Operational frequency ( $f_s$ )
- Power on reset (POR)
- Place and Route (PR)
- Probability of flip-flop upset ( $P_{DFFSEU \rightarrow SEU}$ )
- Probability of logic masking ( $P_{logic}$ )
- Probability of transient generation ( $P_{gen}$ )
- Probability of transient capture ( $P(f_s)_{SET \rightarrow SEU}$ )
- Probability of transient propagation ( $P_{prop}$ )
- Radiation Effects and Analysis Group (REAG)
- Reprogrammable (RP)
- Single event functional interrupt (SEFI)
- Single event effects (SEEs)
- Single event latch-up (SEL)
- Single event transient (SET)
- Single event upset (SEU)
- Single event upset cross-section ( $\sigma_{SEU}$ )
- Static random access memory (SRAM)
- Static timing analysis (STA)
- System on a chip (SOC)
- Time delay ( $\tau_{dly}$ )
- Total Ionizing Dose (TID)
- Transient width ( $\tau_{width}$ )
- Universal Serial Bus (USB)
- Virtex-5QV (V5QV)
- Windowed Shift Register (WSR)

# Agenda

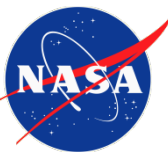
- **Field Programmable Gate Array (FPGA) Devices: Challenges for Critical Applications and Space Radiation Environments**
- **Single Event Upsets (SEUs) and FPGA Configuration**
- **SEUs and Single Event Transients (SETs) in FPGA Data-paths**
- **Fail-Safe Strategies for Critical Applications**



# Motivation: Concerns for using FPGA Devices in Critical Applications



- **Safety**: can life be negatively impacted?
- **Reliability** : will the device operate as expected?
- **Availability**: Includes down-time... is it acceptable?
- **Recoverability**: if the device malfunctions, can the system come back to a working state?  
Destructive?
- **Trust**: Will the insertion of the device compromise security?



# Minimizing Risk: Failure Is Not An Option

- **Critical applications: Scrutiny of design and assurance is a must in order to avoid malfunction or catastrophe.**
  - **Carefully evaluate mitigation... is it applicable?**
  - **Over evaluate limitations/disadvantages/failure modes... everything must be scrutinized... what can go wrong?**
  - **Figure out how to use/implement a solution and what other solutions may be necessary to close gaps of risk.**
- **Question... challenge ... discuss ... absorb ... innovate!**

**Community  
Effort**

***Your solution  
might go further  
than a  
publication***







# FPGA Devices: Challenges for Critical Applications and Space Radiation Environments

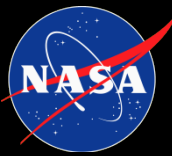




# Protecting A Critical System from Failure

- **Always take into account mission requirements.**
- **Investigate failure modes – evaluate and minimize risk:**
  - Reliability and functional testing (temperature, voltage, mechanical, and logic switching stresses).
  - Radiation testing: Single event effects (SEE), total ionizing dose (TID), and other types.
  - Verification of assurance: analyze test procedures, results, and application to mission solutions.
- **Wisely add mitigation – requirements driven:**
  - Replication with or without correction.
  - Detection:
    - Switch to another device
    - Try to recover state
    - Start over
    - Alert
    - Do nothing...
  - Filtration: e.g., time delay filter (TD),
  - Masking: Protect system operation from failures.

# Investigating Failure Modes: Radiation Testing and SEU Cross Sections



*NEPP: NASA Electronic Parts and Packaging*

**System failures due to SEEs are second order:**

- **Probability that a transistor will change state.**
- **Probability the SEU or SET will cause malfunction.**

**NEPP Heavy-ion testing at Texas A&M University**

## Terminology:

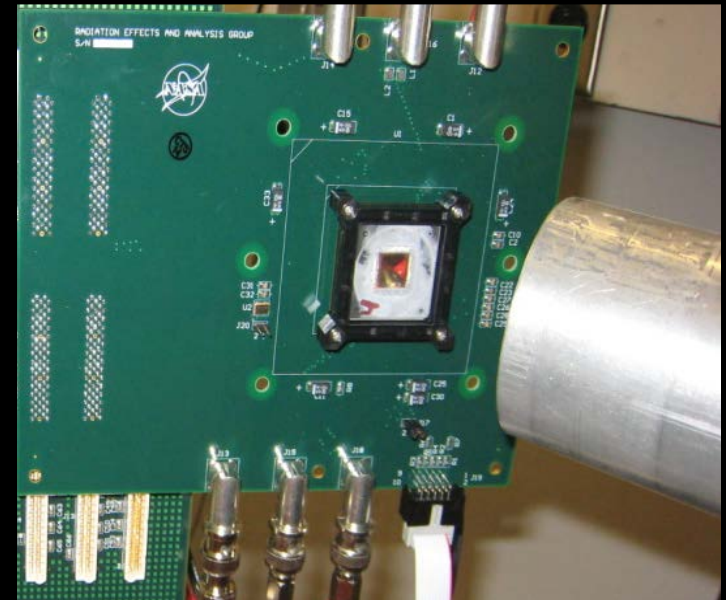
- **Flux: Particles/(sec-cm<sup>2</sup>)**
- **Fluence: Particles/cm<sup>2</sup>**
- **Linear energy transfer (LET)**

**$\sigma_{\text{seu}}$  are calculated at several LET values (particle spectrum).**

**Mean fluence to failure (MFTF) is the inverse of  $\sigma_{\text{seu}}$ .**

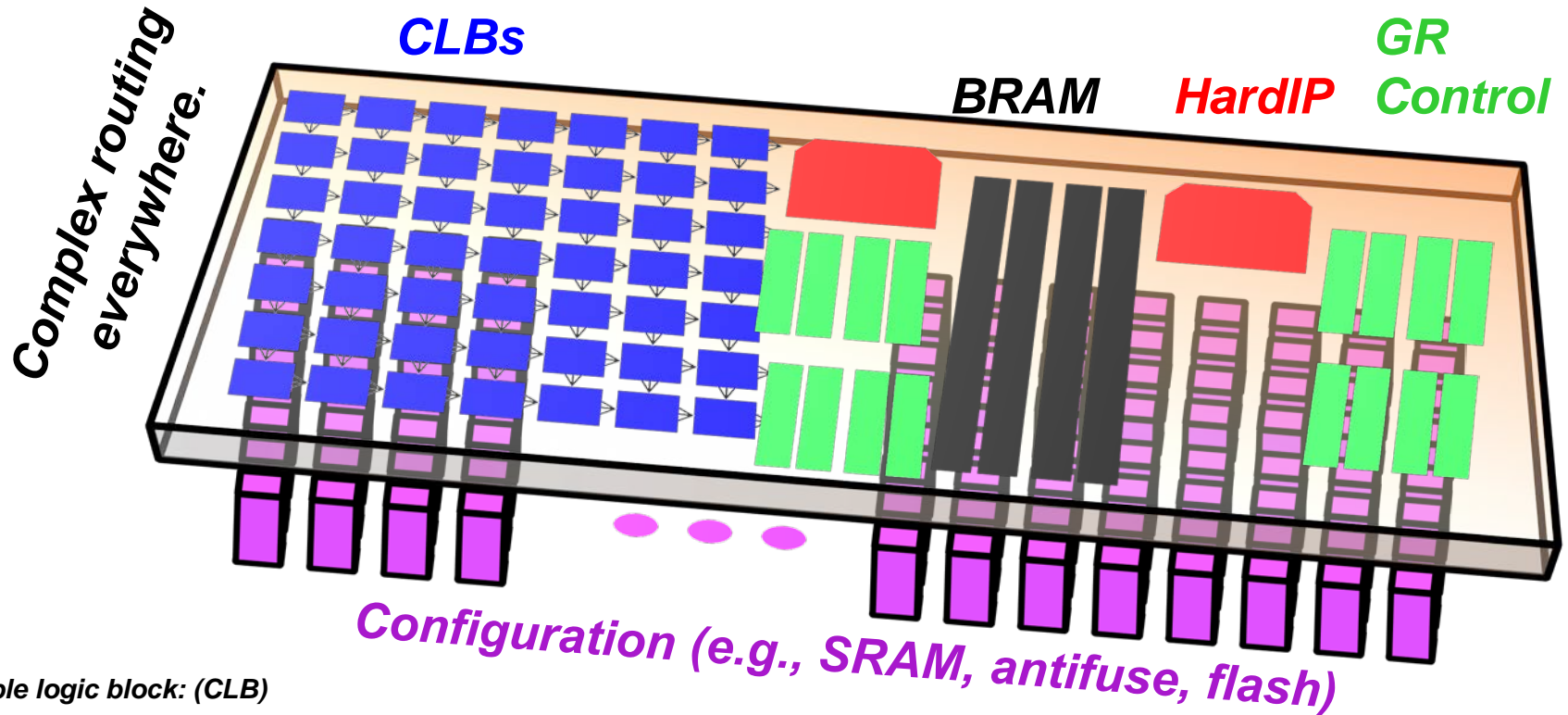
$$\sigma_{\text{seu}} = \frac{\text{\#errors}}{\text{fluence}}$$

$$\text{MFTF} = \frac{1}{\sigma_{\text{seu}}}$$





# FPGA SEU Cross Section Model



Configurable logic block: (CLB)

Block random access memory: (BRAM)

Intellectual property: (IP); e.g., micro processors, digital signal processor blocks (DSP), embedded state machines, etc.

Global Routes: (GR)

Analog circuits

**Device specific and design specific failure modes.**

$$P(fs)_{error} \propto f(P(fs)_{Configuration}, P(fs)_{functionalLogic}, P(fs)_{SEFI})$$

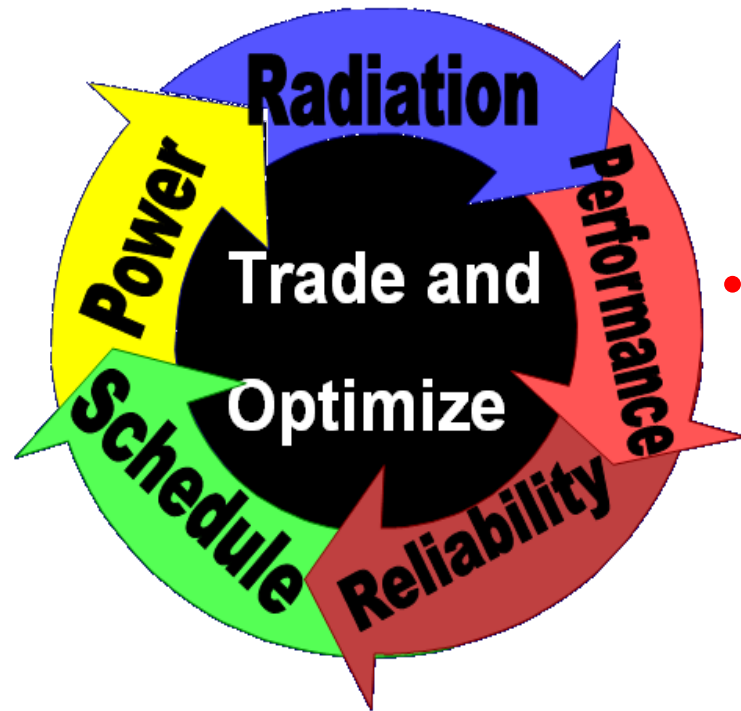
$\sigma_{SEU}$ 
 $\sigma_{SEU}$ 
 $\sigma_{SEU}$ 
 $\sigma_{SEU}$

# Preliminary Design Considerations for Mitigation And Trade Space

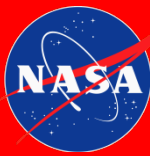


$$P(fS)_{error} = f (P(fS)_{Configuration}, P(fS)_{functionalLogic}, P(fS)_{SEFI} )$$

- Based on mission **requirements** and target device susceptibilities...
  - Does the designer need to add mitigation?
  - How should the designer add mitigation?
  - How can we assure the mitigation?
- Will there be compromises?
  - Performance and speed
  - Power
  - Schedule
  - Mitigating the susceptible components?
  - Reliability (working and mitigating as expected)?



***Security is now an additional component in the trade space.***



**Verify Applied Mitigation and Protection:  
THIS IS CHALLENGING!**

**Theoretical assumptions and modeling do  
not always match reality...**

**Too many unknowns to model**

# Single Event Upsets and FPGA Configuration

$$P_{\text{configuration}} + P(fs)_{\text{functionalLogic}} + P_{SEFI}$$

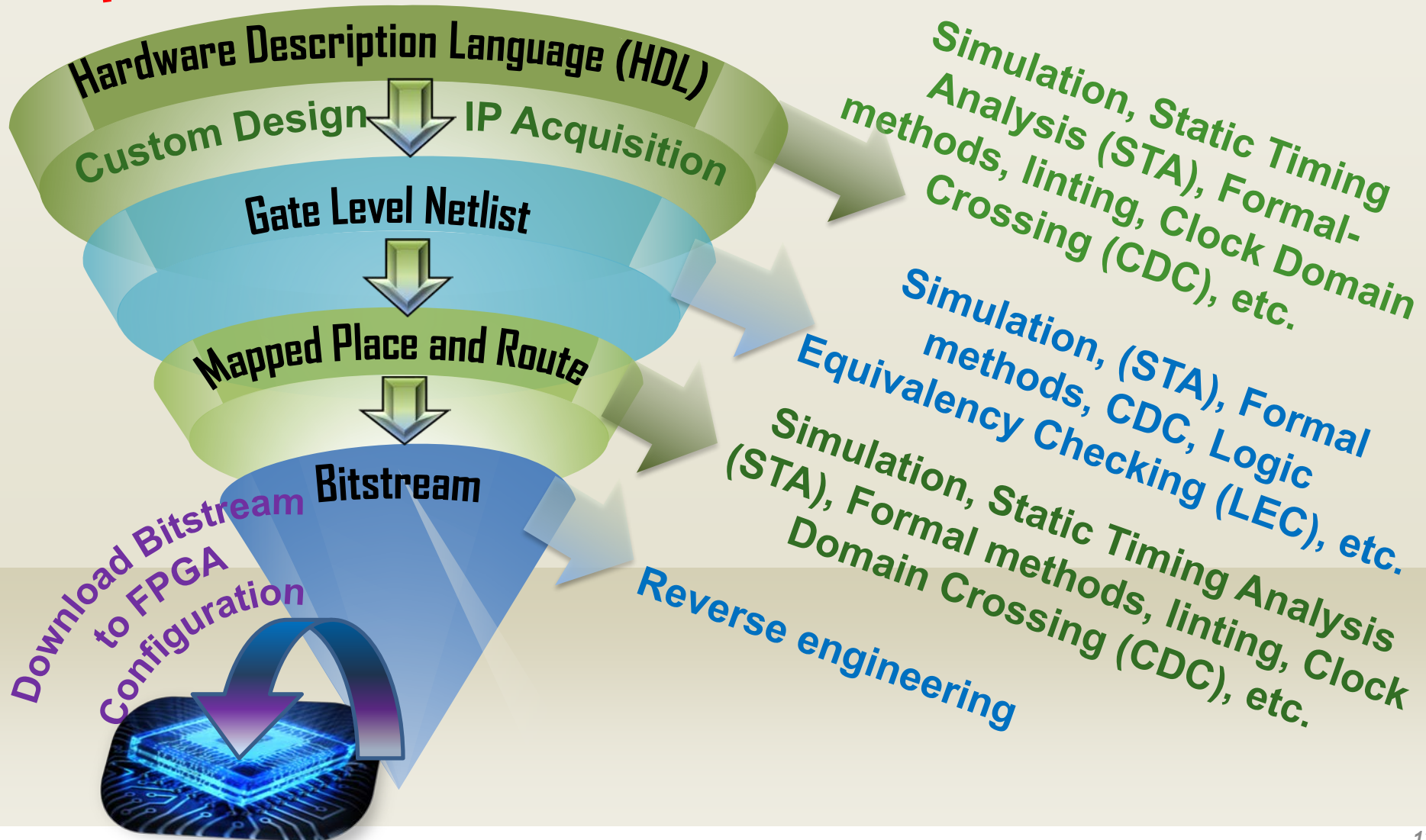




# FPGA Design Process and Configuration Creation



## Product Requirements





# FPGA Devices Are Defined by Their Configuration Type

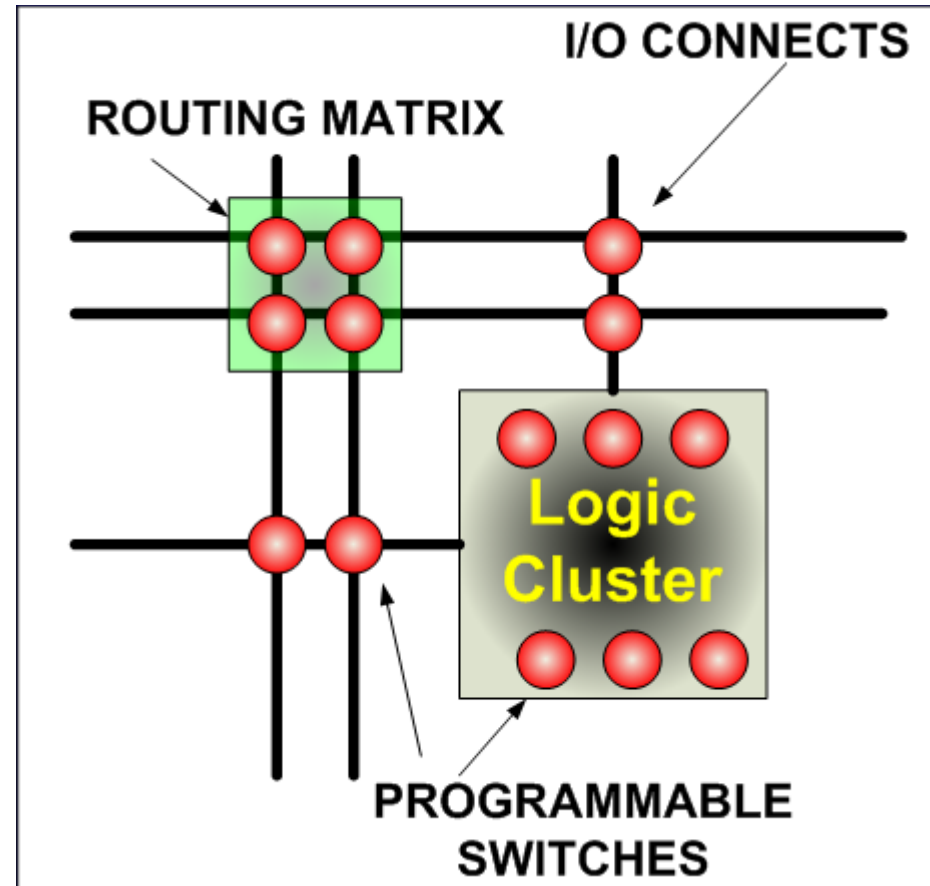


- **Configuration Defines:** Arrangement of pre-existing logic via programmable switches

- Functionality (logic cluster)
- Connectivity (routes)

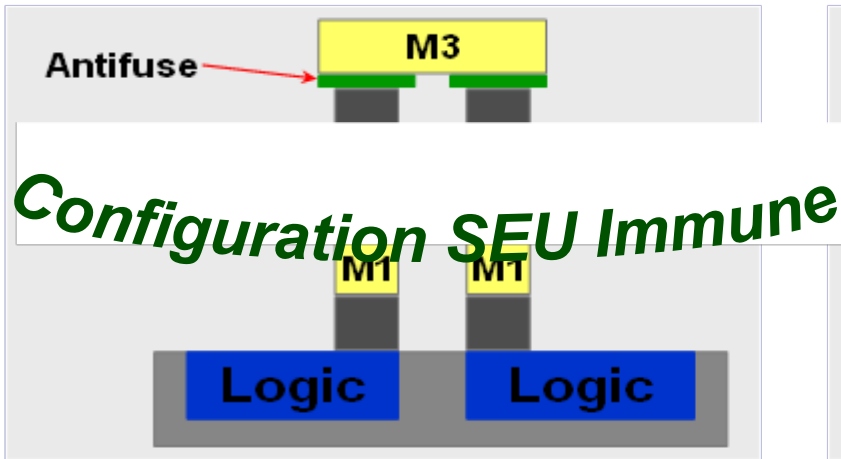
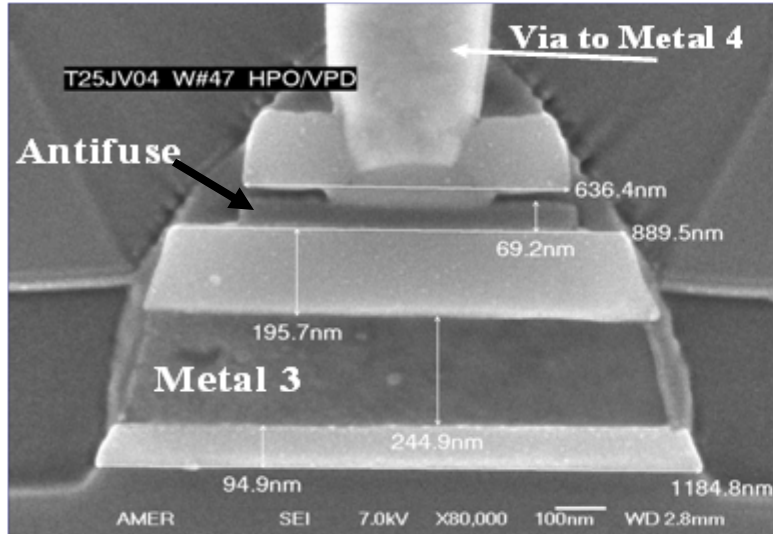
- **Programming Switch Types:**

- **Antifuse:** One time Programmable
- **SRAM:** Reprogrammable
- **Flash:** Reprogrammable

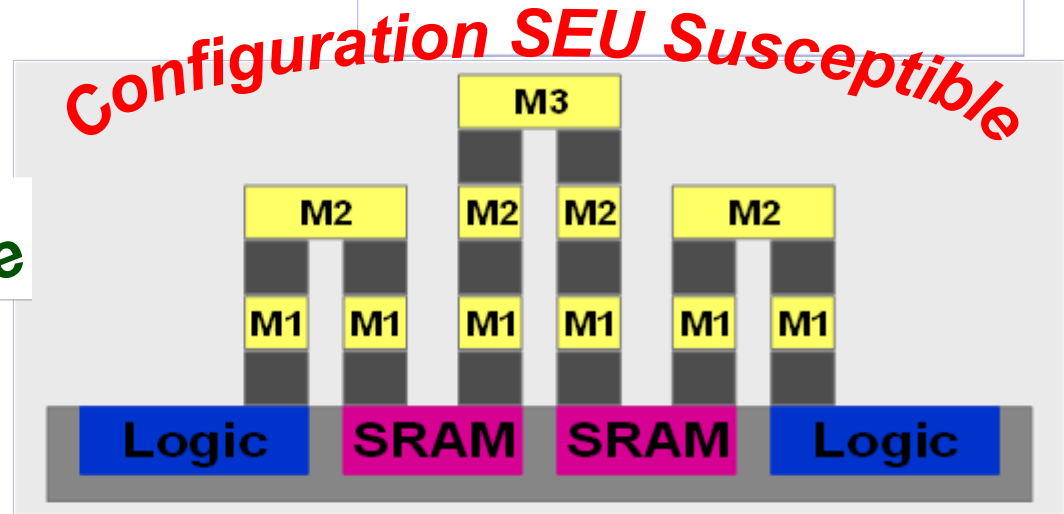
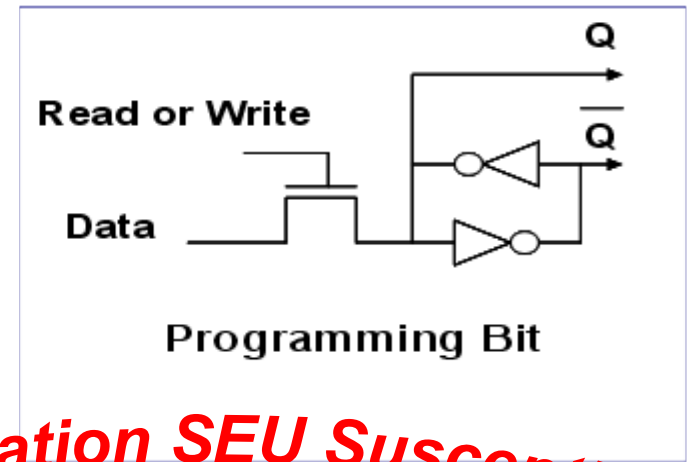


# FPGA Configuration Implementation and SEU Susceptibility

## ANTIFUSE (one time programmable)



## SRAM (reprogrammable)





# Configuration SEU Test Results and the REAG FPGA SEU Model

*Table shows the most significant SEE responses during accelerated radiation testing.*

$$P(fs)_{error} \propto P(fs)_{Configuration} + P(fs)_{functionalLogic} + P(fs)_{SEFI}$$

Configuration Type	REAG Model $P(fs)_{error}$
Antifuse	$P(fs)_{functionalLogic} + P(fs)_{SEFI}$
SRAM (non-mitigated)	$P(fs)_{Configuration} + P(fs)_{SEFI}$
Flash	$P(fs)_{functionalLogic} + P(fs)_{SEFI}$
Hardened SRAM	$P(fs)_{configuration} + P(fs)_{functionalLogic} + P(fs)_{SEFI}$ <i>Low</i> (with red arrow pointing to $P(fs)_{configuration}$ )

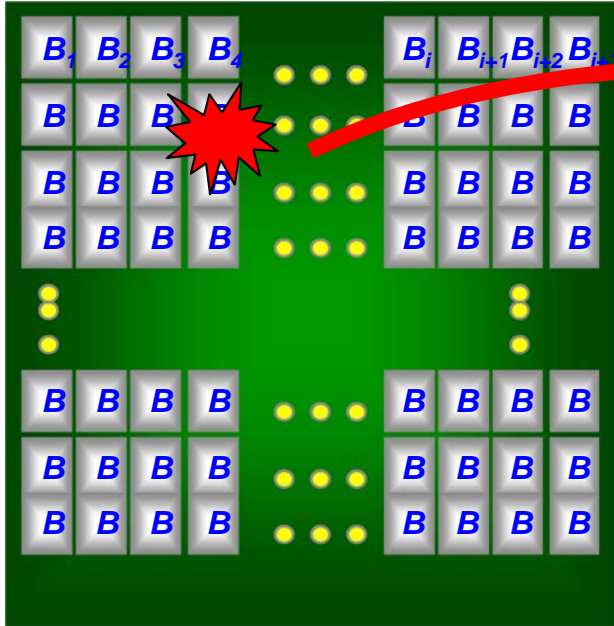


# What Does The Last Slide Mean?

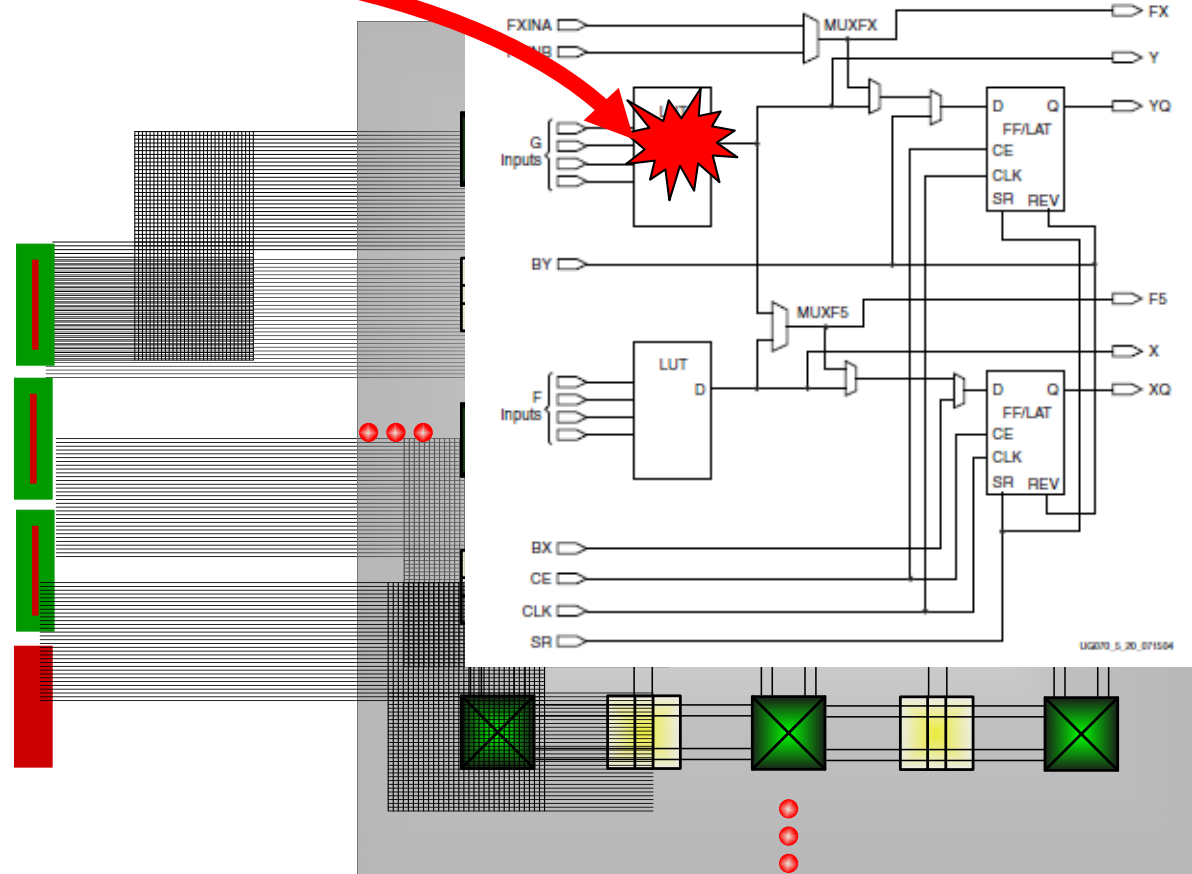
<b>FPGA Configuration Type</b>	<b>Susceptibility</b> Data-path: Combinatorial Logic (CL) and Flip-flops (DFFs); Global: Clocks and Resets; Configuration
Antifuse	Configuration has been designated as hard regarding SEEs. Susceptibilities only exist in the data paths and global routes. However, global routes are hardened and have a low SEU susceptibility.
SRAM (non-mitigated)	Configuration has been designated as the most susceptible portion of circuitry. All other upsets (except for global routes) are too statistically insignificant to take into account. E.g., it is a waste of time to study data path transients, however clock transient studies are significant.
Flash	Configuration has been designated as hard (but NOT immune) regarding SEEs. Susceptibilities also exist in the data paths and global routes (e.g., clocks and resets).
Hardened SRAM	Configuration has been designated as hardened (but NOT hard) regarding SEEs. Susceptibilities also exist in the data paths and global routes (e.g., clocks and resets).

# Take Note: Configuration SRAM is NOT Utilized the Same Way as Traditional SRAM

An affected active/used bit has the ability to instantaneously cause an unexpected effect



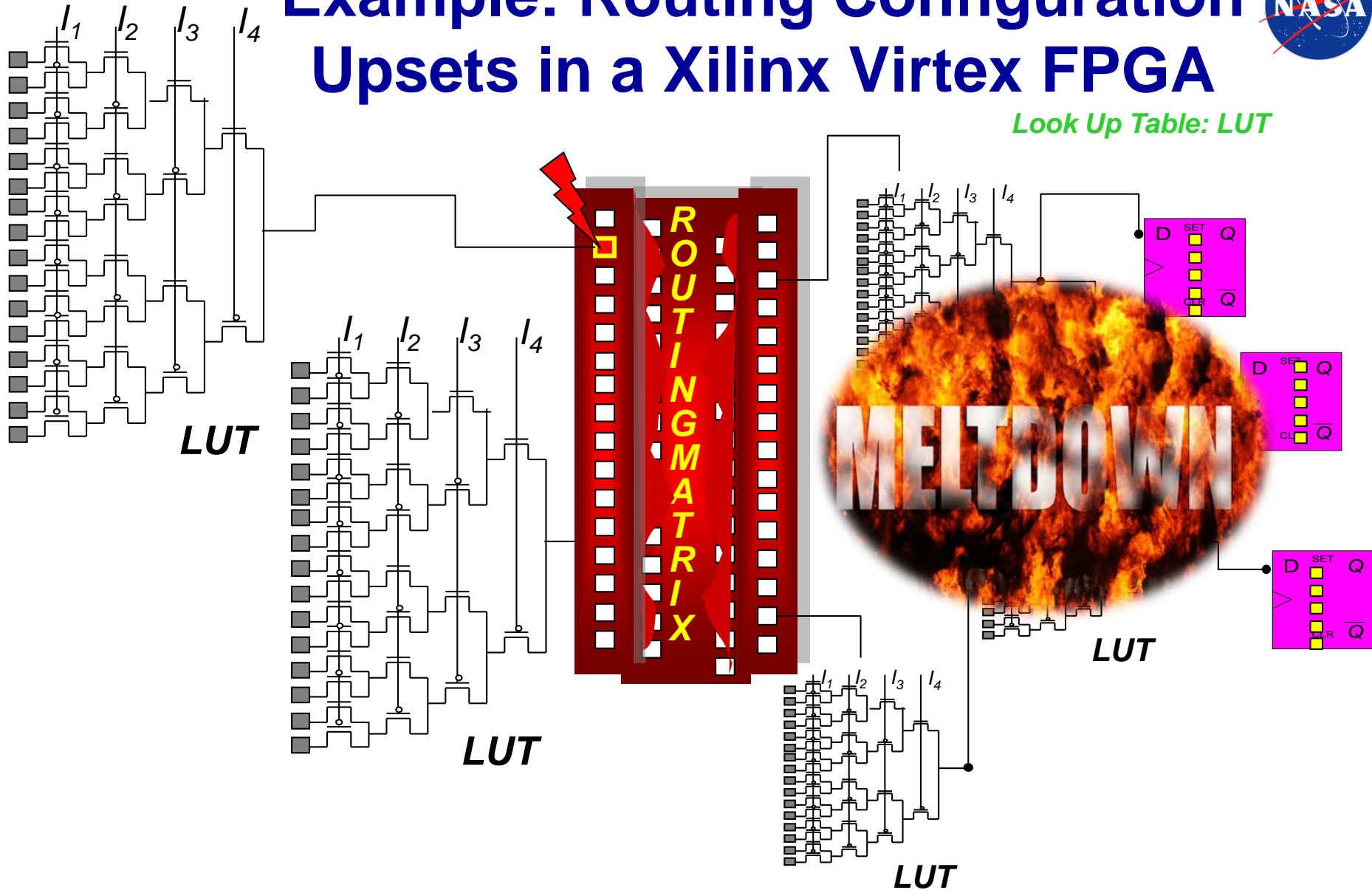
- Direct connections from configuration to user logic.



**No Read-Write cycle required!**



# Example: Routing Configuration Upsets in a Xilinx Virtex FPGA



**One configuration bit flip can cause significant malfunction.  
Mitigate appropriately (Single Event Functional Interrupt (SEFI)).**



# Fixing SRAM-based Configuration...Scrubbing Definition

- We address configuration susceptibility via scrubbing: **Scrubbing is the act of simultaneously writing into FPGA configuration memory as the device's functional logic area is operating with the intent of correcting configuration memory bit errors.**

***Configuration scrubbing only pertains to SRAM-based configuration devices.***



# Scrubbers: Internal versus External

- **Internal and external scrubbers are implemented to correct configuration bit-flips:**
  - **Internal scrubber: is created out of hard cores that reside inside the FPGA device; or is created out of user fabric logic blocks located inside the FPGA device.**
  - **External scrubber is implemented in a separate device .**
- **Typically, external scrubbers are implemented in anti-fuse FPGA or flash-based FPGAs.**
- **Internal scrubbers are more susceptible than external scrubbers.**

# Scrubbers: When Reality Defies Theory



- Internal scrubbers are **expected** to provide satisfactory results in proton and neutron environments.
  - Scrubber clock circuitry are not highly susceptible to protons or neutrons because of their high drive strength.
  - Scrubber should not require a large amount of circuitry.
- **Note: Proton radiation testing of the Intel Cyclone 10 showed the device's internal scrubber does not work as expected.**
  - Scrubber failed to remain operable with a fluence of  $1 \times 10^8$  particles/cm<sup>2</sup> at 100MeV.
  - Results are unexpected.
- **Implementation of the scrubber means everything!**
  - Did Intel use a processor based internal scrubber?
  - Use of memory will cause the scrubber to be more susceptible than expected.
  - Is the scrubber based on single error correct double error detect (SECCDED)... multiple bit upsets will break the scrubber and potentially write bad things into the configuration.



# Scrubbing Warning!

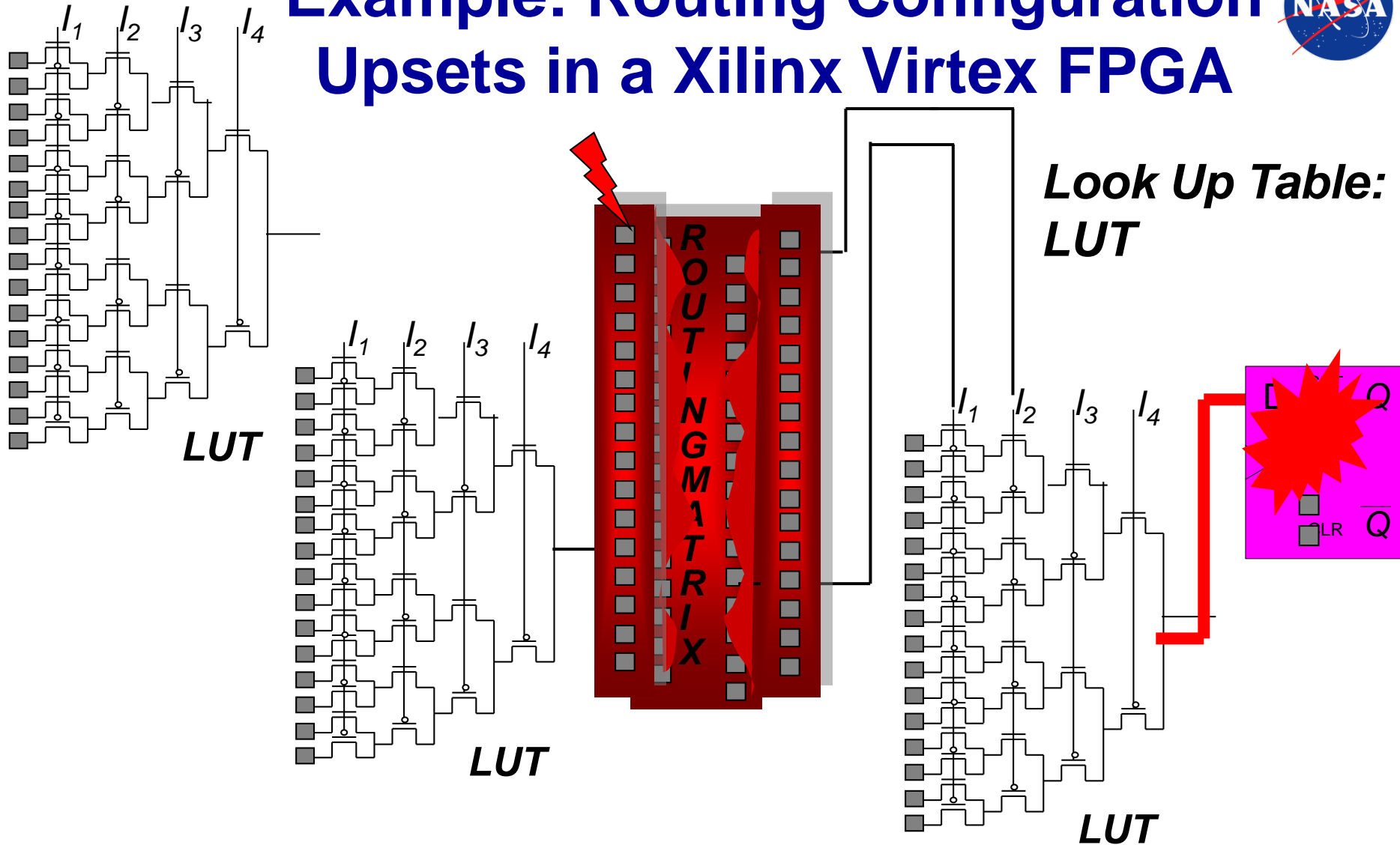
**Correcting a configuration bit does not fix the state in the functional logic path.**

***Reliably getting to an expected state after a configuration-bit SEU (that affects the design's functionality) requires one of the following:***

- Fix configuration bit + (reset or correct DFFs) or***
- Full reconfiguration.***



# Example: Routing Configuration Upsets in a Xilinx Virtex FPGA



**Configuration + design state must be corrected after a configuration SEU hit.**



# Data-path SEU Susceptibility and Analysis : the NASA Electronic Parts and Packaging (NEPP) FPGA Model



Berg M., "FPGA SEE Test Guidelines", NASA Radiation Effects and Analysis Group Website:  
[https://nepp.nasa.gov/files/23779/FPGA\\_Radiation\\_Test\\_Guidelines\\_2012.pdf](https://nepp.nasa.gov/files/23779/FPGA_Radiation_Test_Guidelines_2012.pdf), July 2012.

# Synchronous Design Methodology and SEU Modeling



- Design topology dictates SEU susceptibility
  - Topology: how are components connected and how is data flow controlled:
    - Edge triggered Flip flops (DFFs) versus latches
    - Logic cells: Sea of gates versus look up tables (LUTs)
    - High capacitive routing
    - Large fan-out or large fan-in or feedback paths
- Synchronous design has proven to be the most reliable means of design and is the most common design topology used world wide
  - Makes state transitioning deterministic (reliability matters).
  - Provides distinct boundary points – relates state to clock cycle.
  - Easiest method to verify.
  - Automated design tools are geared for synchronous methodology.

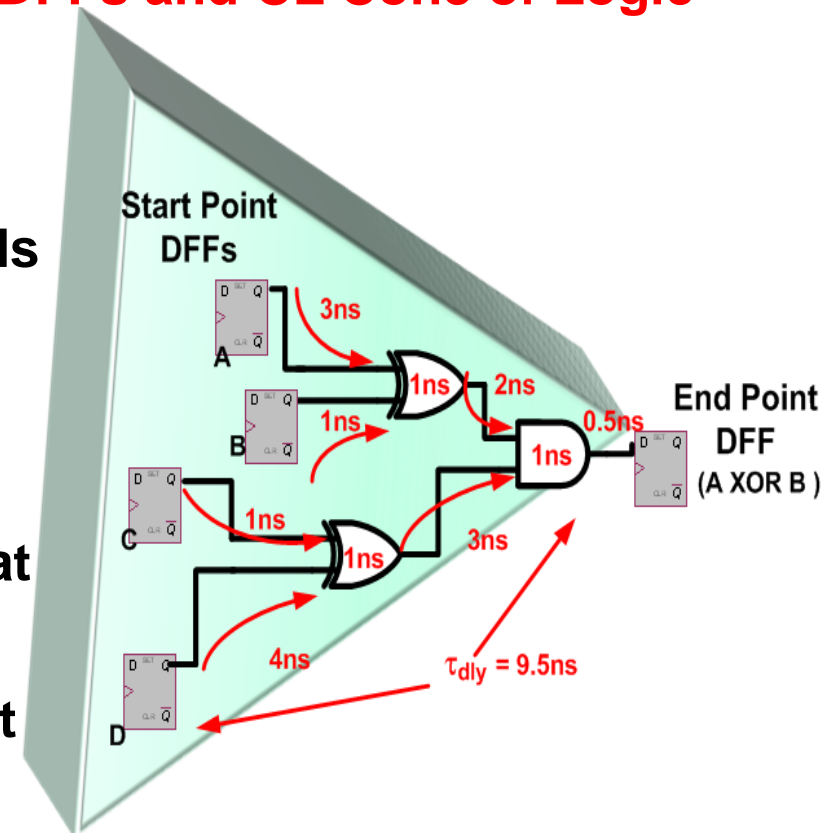
***This presentation pertains to synchronous designs.***

# Synchronous Design Data Path Components



- Designs are comprised of:
  - Combinatorial Logic (CL)
  - Edge Triggered Flip-Flops (DFFs)
- Each DFF has a cone of logic that feeds its input pin (modular approach to system analysis).
  - EndPoint is DFF point of analysis.
  - Startpoints are DFFs (or inputs) that feed the EndPoint.
  - If an EndPoint has feedback, then it is its own StartPoint.
- There is a delay from StartPoint DFF to EndPoint DFF (routes and CL):  $\tau_{dly}$
- $\tau_{dly}$  is CL compute time

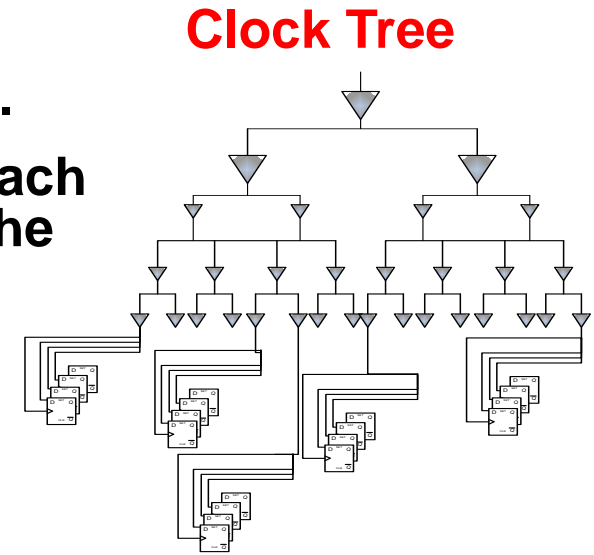
## DFFs and CL Cone of Logic



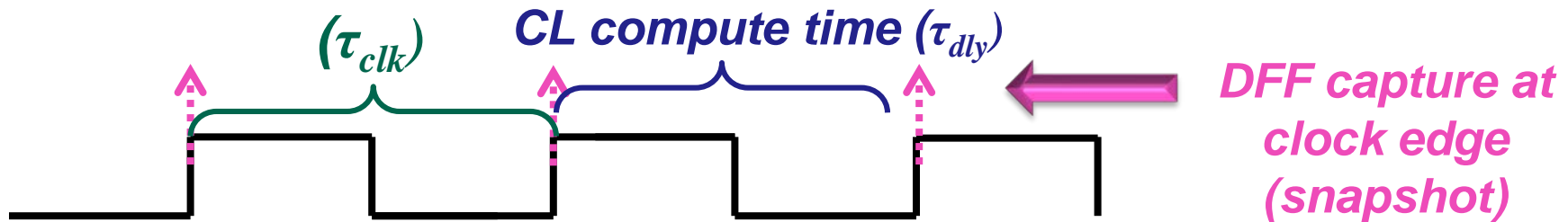
# Synchronous Design Clocks And Data Path Components



- All DFFs are connected to a clock (clock tree).
- Clock tree is a balanced structure such that each DFF will experience a clock edge at virtually the exact moment in time.
- Clock period:  $\tau_{clk}$
- Clock frequency:  $f_s$   $\tau_{clk} = \frac{1}{f_s}$



$$\tau_{dly} < \tau_{clk} - (\text{setup time} + \text{overhead})$$

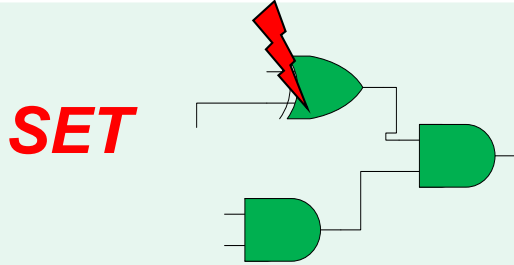
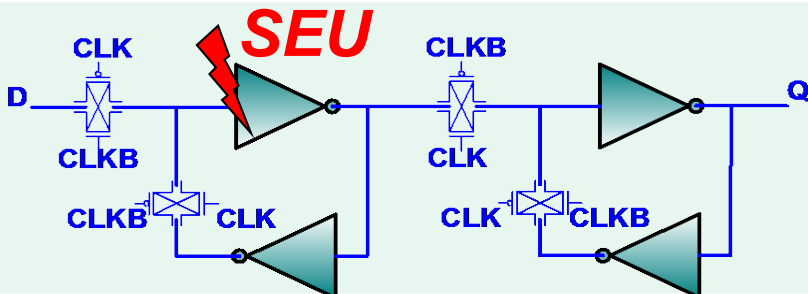




- **Synchronous design: compute and hold in a deterministic manor.**
- **DFF is locked during compute time (its state cannot be affected)**
- **Created to reduce faults due to transistor switching noise.**
- **Less susceptible to SEEs than asynchronous design.**



# SEUs and SETs in Combinatorial Logic and Edge Triggered Flip Flops



Combinatorial (CL)	Sequential (DFF)
<p>Logic function generation (computation)</p>	<p>Captures and holds state of data input at rising edge of clock</p>
 <p><b>SET</b></p>	 <p><b>SEU</b></p>
<p>Glitch in the CL:</p>  <p><b>Double Sided</b></p>	<p>SET Capture in DFF loop</p>  <p><b>Single Sided</b></p>

**Example is an edge triggered DFF. Its effects are significantly different than a latch due to master-slave topology.**

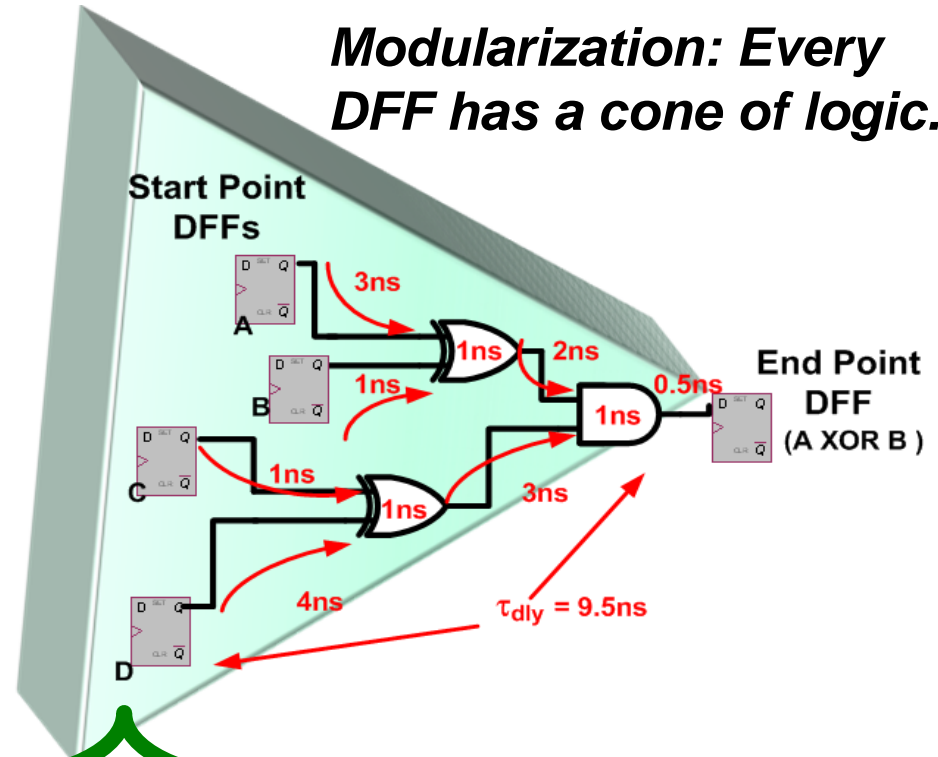
# SEEs and How They Affect Synchronous System Next State



SEEs are asynchronous – most occur somewhere in the clock cycle.

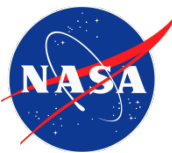
*Modularization: Every DFF has a cone of logic.*

*Question? If a SEU or a SET occurs, will it cause the next state of the system be incorrect?*

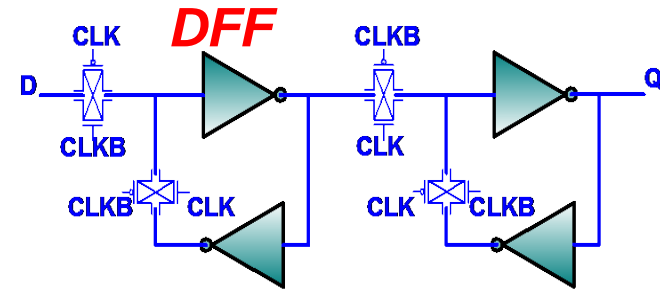


$\nabla$  **EndPoint DFF SEUs** + **StartPoint DFF SEUs** + **CL SETs**  
*EndPoint DFF* **DFF upsets that occur at the clock edge.** **DFF upsets that occur between clock edges and are captured by EndPoints.** **Single Event Transients captured by EndPoints.**  
**Internal DFF transient gets latched.** **Internal DFF latch flips state.**

# FPGA SEU Model Data Path Components



- DFF SEU: **Generation** in DFF ( $P_{DFFSEU}$ )

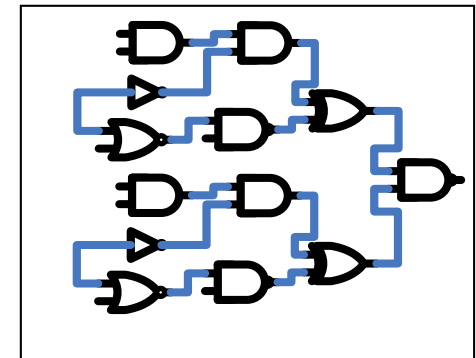


- StartPoint DFF SEU Capture  $P(fs)_{DFFSEU \rightarrow SEU}$ :

- Logic **Masking** ( $P_{logic}$ )
- **Capture**: DFFs are masked from what happens during compute time. Did the StartPoint SEU occur early enough in the clock cycle for the EndPoint to capture the upset? Is the StartPoint Logically masked from the EndPoint?

- CL SET Capture  $P(fs)_{SET \rightarrow SEU}$ :

- SET **Generation** ( $P_{gen}$ )
- SET **Propagation** strength ( $P_{prop}$ )
- Logic **Masking** ( $P_{logic}$ )
- **Capture**: Can the SET propagate without being masked to reach the EndPoint DFF at the next clock edge?



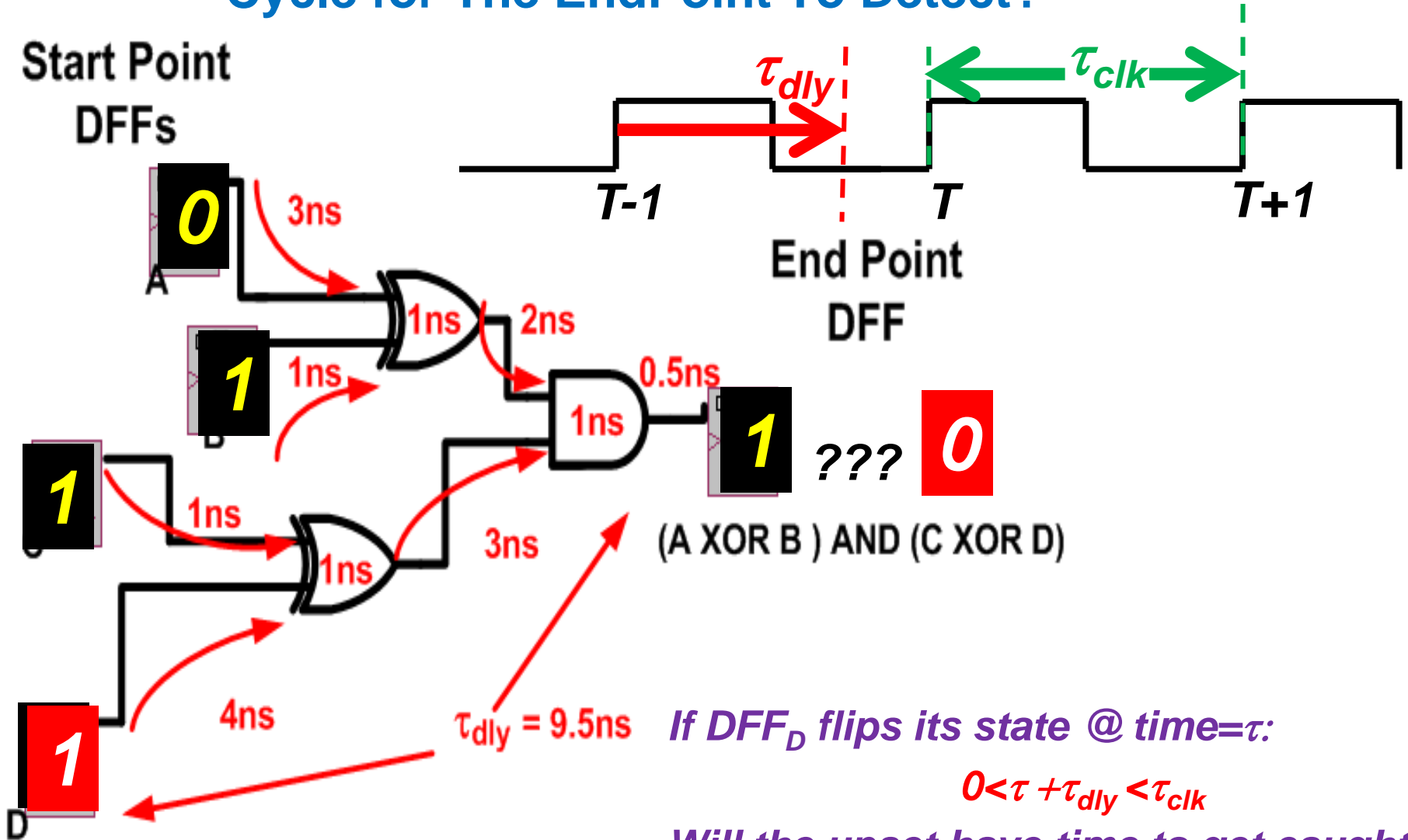
CL



# StartPoint SEUs And System Next State:

## Does the StartPoint Flip Early Enough In The Clock Cycle for The EndPoint To Detect?

Start Point  
DFFs



*If DFF<sub>D</sub> flips its state @ time= $\tau$ :*

$$0 < \tau + \tau_{dly} < \tau_{clk}$$

*Will the upset have time to get caught?*

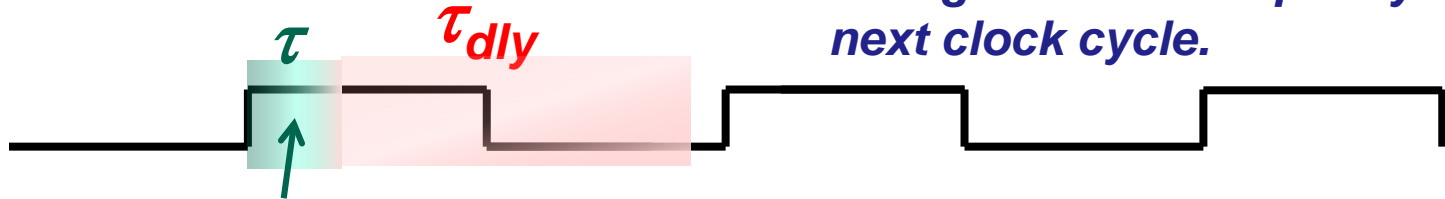


# Potential For A StartPoint SEU To Affect Its EndPoint Next State (Temporal Masking)

*StartPoint SEU occurs at a point in time within a clock cycle*

$$\tau + \tau_{dly} < \tau_{clk}$$

$\tau_{dly}$  (Delay from StartPoint to the EndPoint) is fixed (road block). EndPoint will only be affected if change reaches its pin by the next clock cycle.



*Only portion of clock cycle that StartPoint can flip and affect next state*

$$\frac{\tau}{\tau_{clk}} < \frac{\tau_{clk} - \tau_{dly}}{\tau_{clk}} = 1 - \frac{\tau_{dly}}{\tau_{clk}}$$

*Probability for EndPoint to detect flipped StartPoint at clock edge (next state). The path delay bounds probability of capture.*

$$\tau \text{ fs} < 1 - \tau_{dly} \text{ fs}$$

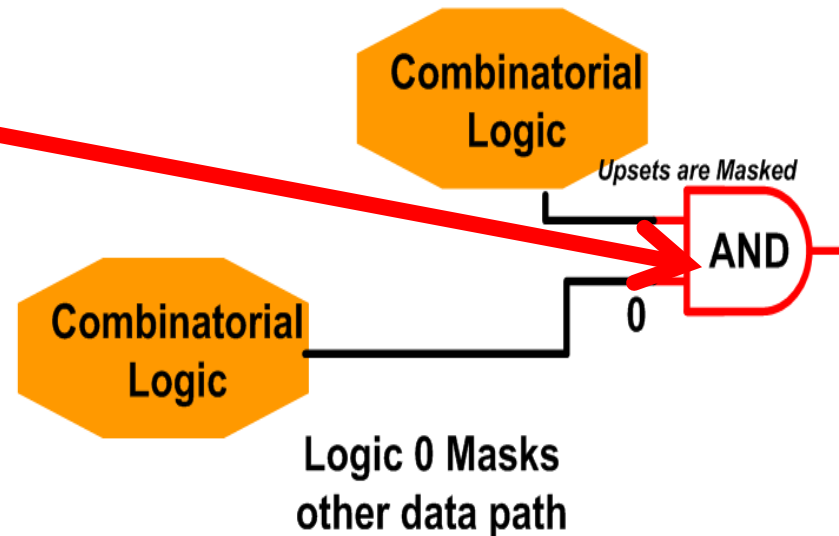
**The probability that a StartPoint DFF SEU will affect the system next state is inversely proportional to system frequency.**

# Logic Masking: $P_{logic}$

$P_{logic}$ : Probability that an SET can logically propagate through a cone of logic. Based on state of the combinatorial logic gates and their potential masking.

“AND” gate reduces probability that SET will logically propagate

$$0 < P_{logic} < 1$$



- Determining  $P_{logic}$  for a complex system can be very difficult in real applications.
- Simulation and fault injection will only provide a glimpse of the state space; and is not sufficient to determine  $P_{logic}$ .

# Details of Capturing StartPoint DFFs

$$\forall_{DFF} \left( \sum_{j=1}^{\#StartPoint DFFs} \beta P(fs)_{DFFSEU(j)} (1 - \tau_{dly(j)} fs) P_{logic(j)} \right)$$

Upset generated internally to DFF between clock edges
Design Topology and Temporal Masking
Logic masking

- SEU generation occurs in a StartPoint between rising clock edges ( $\beta P(fs)_{DFFSEU}$ )
- Design topology and temporal effects ( $\tau_{dly}$ ):
  - Increase path delay – decrease probability of capture
  - Increase frequency – decrease probability of capture
- StartPoint upsets can be masked by logic between the StartPoint and its EndPoint ( $P_{logic}$ )

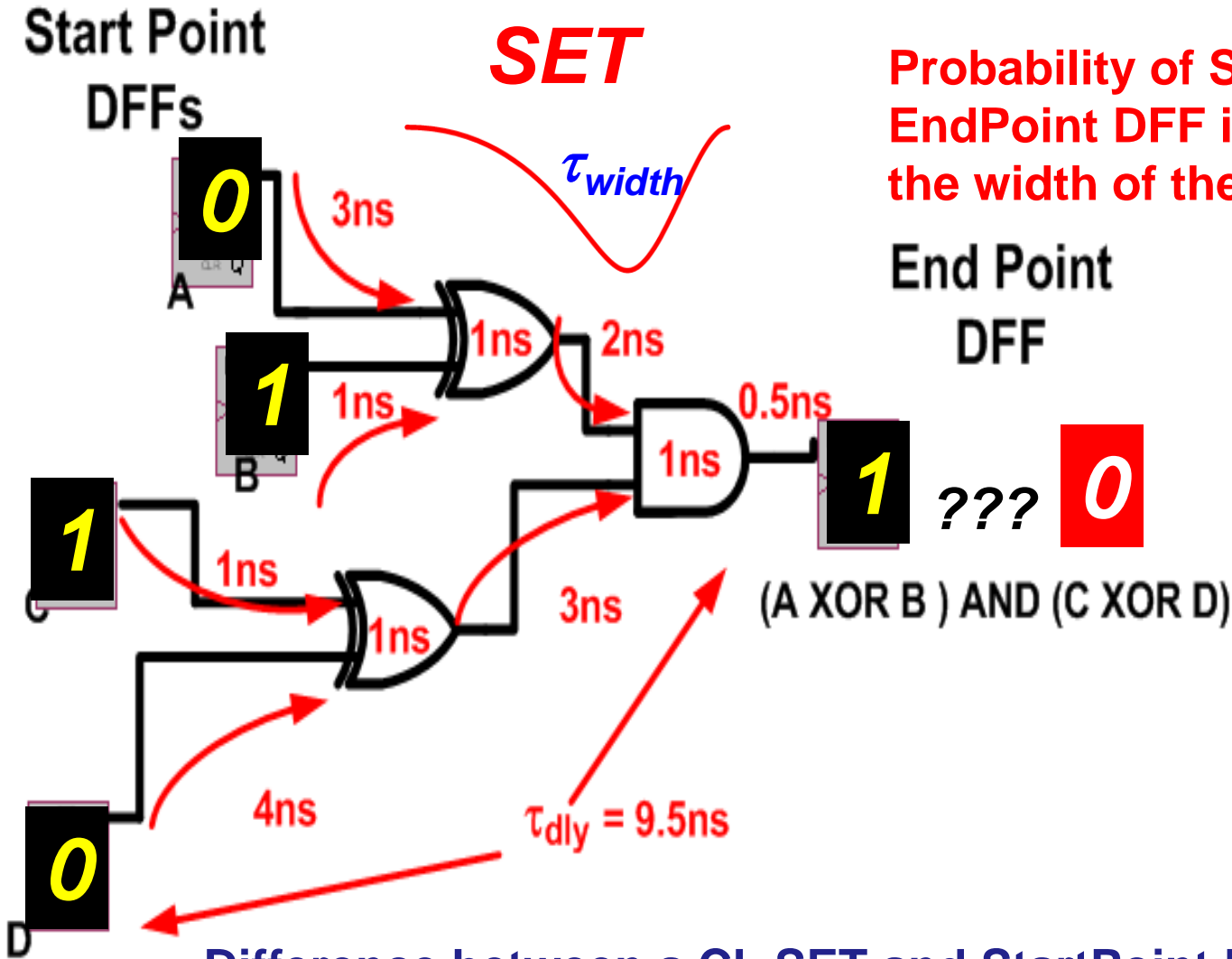




# Synchronous System: CL SET Capture

**SET**

Probability of SET capture by EndPoint DFF is proportional to the width of the SET.



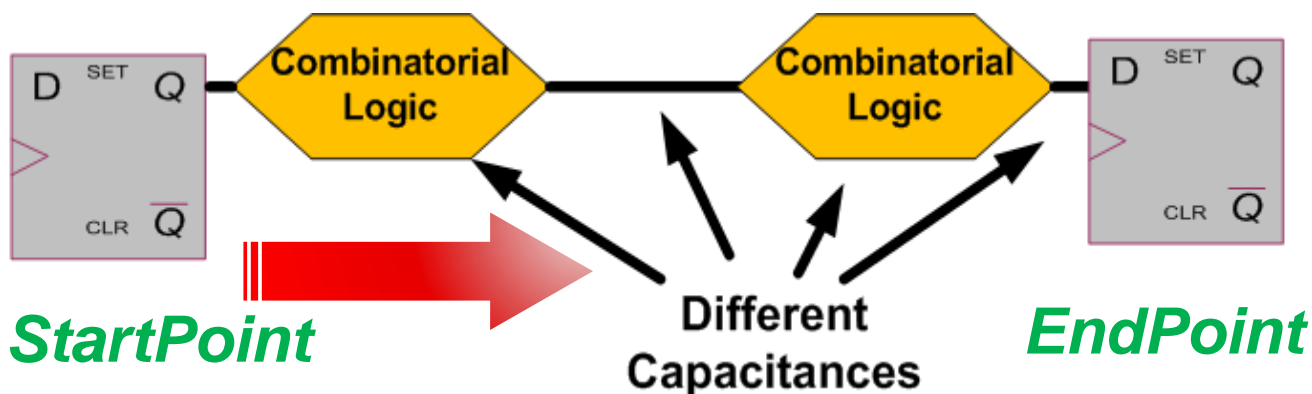
End Point DFF

$$(A \text{ XOR } B) \text{ AND } (C \text{ XOR } D)$$

Difference between a CL SET and StartPoint DFF-SEU:  
Double sided glitch versus single sided state switch.

# SET Propagation to an EndPoint DFF: $P_{prop}$

- In order for the data path SET to become an upset, it must propagate and be captured by its Endpoint DFF.
- $P_{prop}$  only pertains to electrical medium (resistance and capacitance (RC) of path)
  - RC can cause SET amplitude reshaping
  - RC can cause SET width reshaping
  - RC can cause SET oscillation
- Small SETs or paths with high RC have low  $P_{prop}$
- Depends on LET – be careful of fault injection results – they do not take into account the correlation between LET and SET strength (size).



# Details of CL SET Capture in a Synchronous System: $P(fs)_{DFF \rightarrow SET}$

$$\forall_{DFF} \left( \sum_{i=1}^{\#CombinatorialCells} (P_{gen(i)} P_{prop(i)} P_{logic} \tau_{width(i)} fs) \right)$$



SET is Generated

SET logic masking

SET can propagate through electrical medium (routes and gates) and reach the End-Point

Width of SET relative to clock period

$P(fs)_{DFF \rightarrow SET}$  is proportional to  $\tau_{width} / \tau_{clk}$  or  $\tau_{width} fs$

- Note: difference in probability analysis between CL SET and StartPoint SEU is due to double edge function versus single sided.
- Increase frequency – increase probability of capture.
- Increase CL – increase probability of capture?? **Might create more masking (error detection and correction is the perfect example).**
- **Increase LET – increase the width of the SET.**

# NEPP FPGA Model: Putting it All Together



## ... Analyzed Per Particle LET

$$\sum_{k=1}^{\#EndPoint_{DFFs}} P_{logic(k)} * \left( \begin{array}{l} \alpha P(fS)_{DFFSEU(k)} + \\ \sum_{j=1}^{\#StartPoint_{DFFs}} ( \beta P(fs)_{DFFSEU(j)} (1 - \tau_{dly(j)} fS) ) * P_{logic(j)} + \\ \sum_{i=1}^{\#CL} ( P_{gen(i)} * P_{prop(i)} * P_{logic(i)} * \tau_{width(i)} fS ) \end{array} \right)$$

- Model is not expected to qualify a design ( $P_{logic}$  is too difficult to predict).
- Model is expected to assist in data analysis (clarifies events).
- Can determine if DFFs are more dominant than SETs:
  - Indirectly proportional to frequency – then DFFs are dominant.
  - Directly proportional to frequency – then SETs are dominant.
- Same philosophy can be used to determine mitigation strength.



# NEPP FPGA Model: Mitigation...

## Analyzed Per Particle LET

$$\sum_{k=1}^{\#EndPoint_{DFFs}} \left( \begin{array}{l} \text{EndPoint} \\ \text{Logic} \\ \text{Masking} \\ P_{logic(k)} * \end{array} \left( \begin{array}{l} \text{EndPoint} \\ \alpha P(fs)_{DFFSEU(k)} + \\ \text{StartPoints} \\ \sum_{j=1}^{\#StartPoint_{DFFs}} \left( \beta P(fs)_{DFFSEU(j)} (1 - \tau_{dly(j)} fs) \right) * P_{logic(j)} + \\ \text{CL} \\ \sum_{i=1}^{\#CL} \left( P_{gen(i)} * P_{prop(i)} * P_{logic(i)} * \tau_{width(i)} fs \right) \end{array} \right) \right)$$

- $P_{logic}$  can be implemented by a designer for SEE mitigation.
- $P_{prop}$  and  $P_{gen}$  can mitigate SEEs by process, technology geometries, or user placement.
- $P_{logic}$  manipulation is the most common method of mitigation implementation (triple modular redundancy, disable, error detection and correction (EDAC). Deterministic mitigation is essential.
- Note– simply increasing the number of DFFs or increasing CL does not mean you have increased your susceptibility (because of  $P_{logic}$ )... **current prediction methodologies fail us!**



# Warning: Clock Trees and SETs

- **Examples only considered data paths.**
- **However, clock and reset trees (global routes) are susceptible to SETs.**
- **Clock trees in ASICs and FPGAs are the most overlooked mechanism of failure due to ionization.**
- **Global route susceptibilities must be taken into account when determining system risk.**
- **Global route susceptibilities are different for each FPGA device.**

***There is not much a user can do to mitigate clock tree SETs. However, it is imperative to know susceptibilities – probability of occurrence and associated error signatures.***

# Fail-safe Strategies for Data-Path Single Event Upsets (SEUs)



- The following slides will demonstrate commonly used mitigation strategies for FPGA devices.
- What you should learn:
  - The differences between mitigation strategies.
  - Strengths and weaknesses of various strategies.
  - Questions to ask or considerations to make when evaluating mitigation schemes.
  - Which mitigation schemes are best for various types of FPGA devices.
- The scope of this presentation will cover fail-safe strategies for configuration and data-path SEUs



# Fail-Safe Strategies for FPGA Critical Applications

**Goal for critical applications:  
Limit the probability of system error propagation and/or provide detection-recovery mechanisms via fail-safe strategies.**





# Differentiating Fail-Safe Strategies:

- **Detection:**
  - Watchdog (state or logic monitoring).
  - Can range from simplistic checking to complex Decoding.
  - Action (alerting, correction, or recovery).
- **Masking (does not mean correction):**
  - Preventing error propagation to other logic.
  - Requires redundancy + mitigation or detection.
  - Turn off faulty path.
- **Correction (error may not be masked):**
  - Error state (memory) is changed/fixed.
  - Need feedback or new data flush cycle.
- **Recovery:**
  - Bring system to a deterministic state.
  - Might include correction.



# Redundancy Is Not Enough

- Simply adding redundancy to a system is not enough to assume that the system is well protected.
- Questions/Concerns that must be addressed for a critical system expecting redundancy to cure all (or most):
  - Define system failure... what is tolerable and what is not.
  - How does the system recover from SEE?
  - How is redundancy implemented?
  - What portions of your system are protected? Does the protection comply with the results from radiation testing?
  - Is detection of malfunction required to switch to a redundant system or to recover?
  - If detection is necessary, how quickly can the detection be performed and responded to?
  - Is detection enough?... Does the system require correction?

***Listed are crucial concerns that should be addressed at design reviews and prior to design implementation.***

# Embedded Mitigation versus User Inserted Mitigation





# Radiation Hardened (per SEU) versus Commercial FPGA Devices

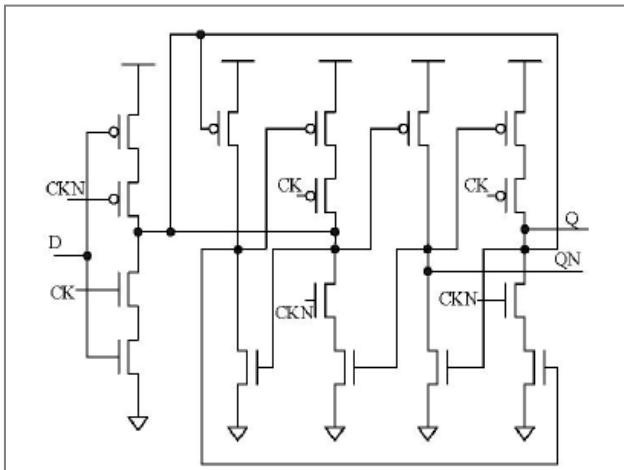


- For this presentation, a radiation hardened (per SEU) device is a device that has embedded mitigation (implemented by FPGA manufacturer – not the user).
- Radiation hardened FPGA devices are available to users. They make the design cycle much easier!
- SEU mitigation is generally applied to the following:
  - **Data-path elements:**
    - Localized redundancy inserted into library cell flip-flops (DFFs).
      - Localized Triple Modular Redundancy (LTMR) or
      - Dual interlocked Cell (DICE)
    - SET filters inserted on the DFF data input pin.
    - SET filters inserted on the DFF clock input pin.
  - **Global routes.**
  - **Memory cells.**

# Localized Redundancy Embedded in Manufacturer DFF Cells

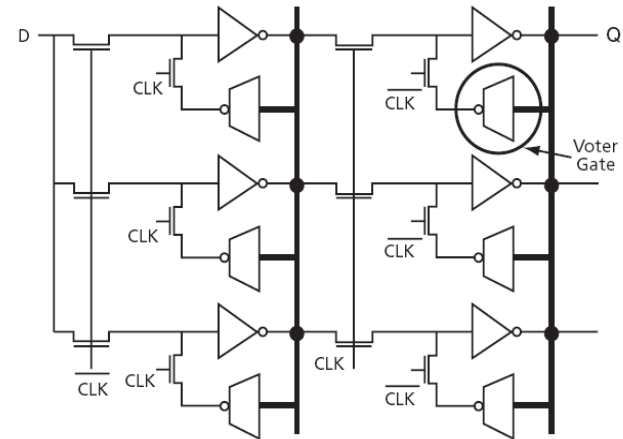
**Warning!** These figures are simplified schematics of the actual implementation.

## Dual Interlocked Cell (DICE)



**Xilinx**

## Localized Triple Modular Redundancy (LTMR)



**Microsemi**

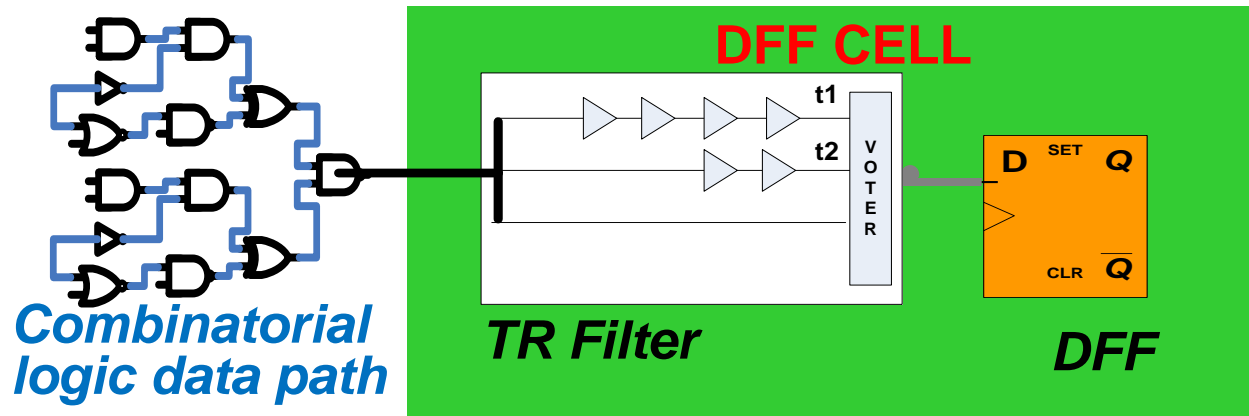
**Problem!** Although DFFs are protected, SETs from the combinatorial logic in the data path and SETs in the global routes can cause incorrect data to be captured by the DFF.

# Embedded Temporal Redundancy (TR): SET Filtration in The Data Path

- Temporal Filter placed directly before DFF.
- Localized scheme that reduces SET capture in the data path.
- Delays must be well controlled.
  - Every delay path shall consistently have a predefined delay and must be verified.
  - Remember: critical applications require deterministic mitigation.
- **Do not implement TR as a user inserted mitigation scheme. Delay must be deterministic and it is too difficult to manage with place and route tools (for real applications).**
- Maximum Clock frequency is reduced by the amount of new delay.

*Crude example of TR implementation*

**Embedded cell implementation is ok. User fabric implementation is not ok.**

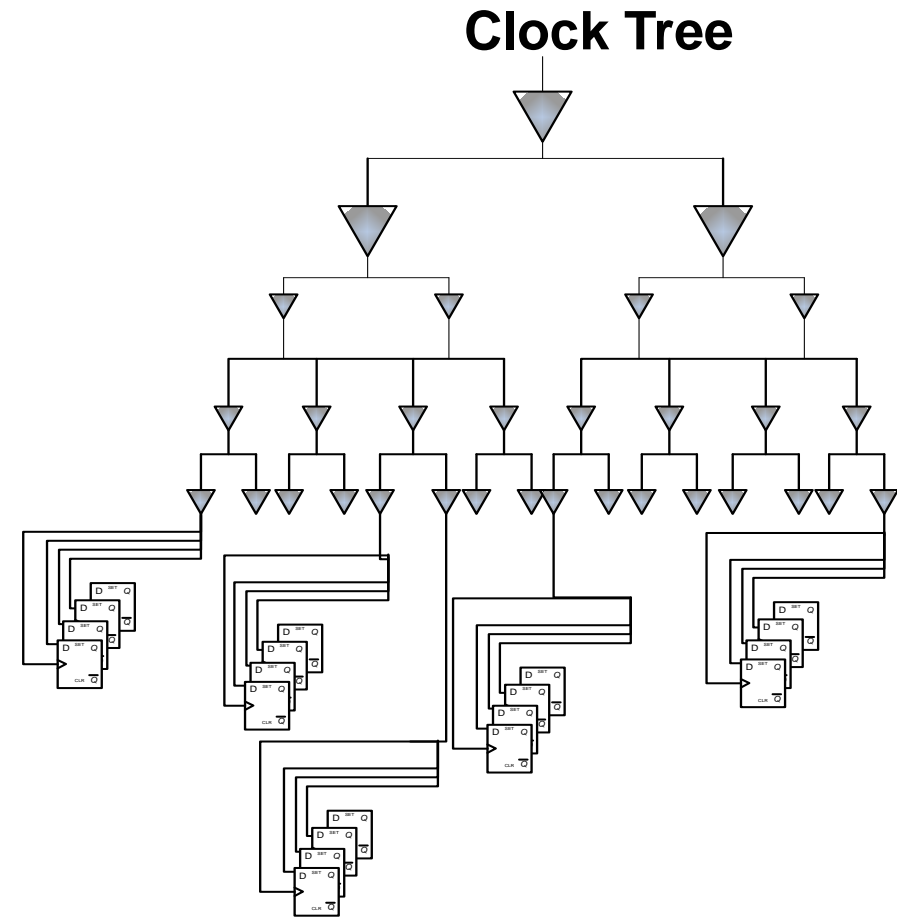




# Embedded Radiation Hardened Global Routes: SET Filtration in The Global Route Path



- Some FPGAs contain radiation-hardened clock trees and other global routes (**Microsemi products only**).
- Global structures are generally hardened by using larger buffers.
- TR has also been used on the DFF clock pin... (**Xilinx V5QV only**).



***Global route susceptibility is often overlooked. Beware, many devices do not have hardened global routes.***

# FPGA Devices and Manufacturer Embedded Mitigation



***DFF: flip flop***

***DICE: Dual interlocked Cell***

Configuration Type	Short List of Device Families	Embedded Mitigation	Most Susceptible Components
SRAM	Stratix, Virtex, Kintex	No	Configuration and clock trees
Antifuse	RTAX, RTSXS	DFFs and clocks (configuration is already hardened by nature)	Combinatorial logic (however susceptibility considered low)
Flash	ProASIC3, RTG4, SmartFusion(2)	Configuration is already hardened by nature.	ProASIC3 and SmartFusion: DFFs and clocks; RTG4: clocks and SETs
Hardened SRAM	Virtex V5QV	Configuration + DICE DFFs + SET filters	Clocks. In some cases additional mitigation may be necessary for configuration and DFFs

***Go to <http://radhome.gsfc.nasa.gov>, manufacturer websites, and other space agency sites for more information on SEU data and total ionizing dose data.***

# User Inserted Mitigation: Flushing, Dual Redundancy, Cold Sparing, and Triple Modular Redundancy (TMR)





# Most Effective Mitigation Strategies

- **Hire knowledgeable and experienced design/verification engineers. Implementation of design matters.**
- **Understand the target environment and mission requirements.**
  - Reliability
  - Availability
  - Weigh consequences of failure
- **Plan ahead :**
  - How to partition and manage power domains (how often can you power flush, what circuitry are affected during power flush (currently used for micro-latchup).
  - How to efficiently insert mitigation.
  - How to alert.
  - How to synchronize redundant circuits (if necessary).
  - Beware of separate clock domain drift.
- **Use data/information that best suites you: differentiate between un-vetted research and application oriented.**



# Most Commonly Implemented System Level Mitigation:

## Reset or Flush... Keeping It Simple!

- Critical applications require all registers (flip-flops) to be connected to a reset.
- A reset is used to force the system to a known (expected) state in a deterministic time period.
- Requires detection of malfunction; or user controlled maintenance scheme.
- All elements are expected to be able to operate from the reset state. However:
  - For some FPGAs, a reset is not enough. The configuration might also have to be flushed (reconfigure or scrub).
  - Availability is affected.
  - Next state information during event is most likely lost.
  - All must be taken into account when determining the effect of activating a reset in a system.

***Warning: Resets are susceptible to SEEs***



# Dual Redundancy

- **Dual redundant systems cannot correct (roll-back is an exception).**
- **Dual redundant systems are great for detection (watch-dogs).**
- **“Compare and Alert” systems must be highly reliable and verifiable.**
- **Generally not all I/O can be monitored or compared.**
  - **Best used for data calculation and manipulation... easiest to place compares on data buses.**
- **Can run in lockstep or free running. Each have unique advantages and limitations (cons).**



# Cold Sparring: Elongation of System Operation



- One active system and alternate inactive systems.
- Upon active system failure, an inactive system is turned on.
- System operation is able to be elongated after failure.
- However:
  - Availability is affected... there is downtime.
  - Can your system afford the downtime (critical application)?
  - How clean is the system switch over?
  - How long is the system switch over.
- Can the system ping-pong between active and inactive components or is that portion of the system considered dead after failure?
  - Ping-ponging can be used for systems that have a low probability of destructive failures.
  - Ping-ponging can be complex and can affect availability.

**Mostly used for degradation mitigation (no ping-pong)**



# Multiple Flushable Components (Sensor Example)

- *Each sensor captures a frame of data.*
  - *Time-tag each frame of data.*
  - *Central unit processes and organizes frames.*
  - *Synchronization signal to start frames.*
  - *Synchronization is challenging... clock skew or system drift.*
- 
- *If one or more components fall out (fail), then synchronize on next frame (not always easy).*
  - *Must strategize for bulk failures.*



# Partial Mitigation

- **Implementation of mitigation can be limited by:**
  - Gate count
  - Timing
  - Accessibility to logic (e.g. IP cores)
  - Efficacy of tool
- **How do we handle the challenge?**
  - Research world: build tools to perform partial mitigation
  - Critical application world: requirements dictate that all inserted mitigation must be proven protection. Currently, partial mitigation is driven by requirements and noted criticality of components regarding mission success.
- **Having an automated tool decide where to put protection is risky:**
  - Tools are not ready to handle the abundance of parameters yet.
  - Mitigation becomes difficult to assure after implementation (theory versus reality)... adds to risk factor.



# System versus Design Mitigation

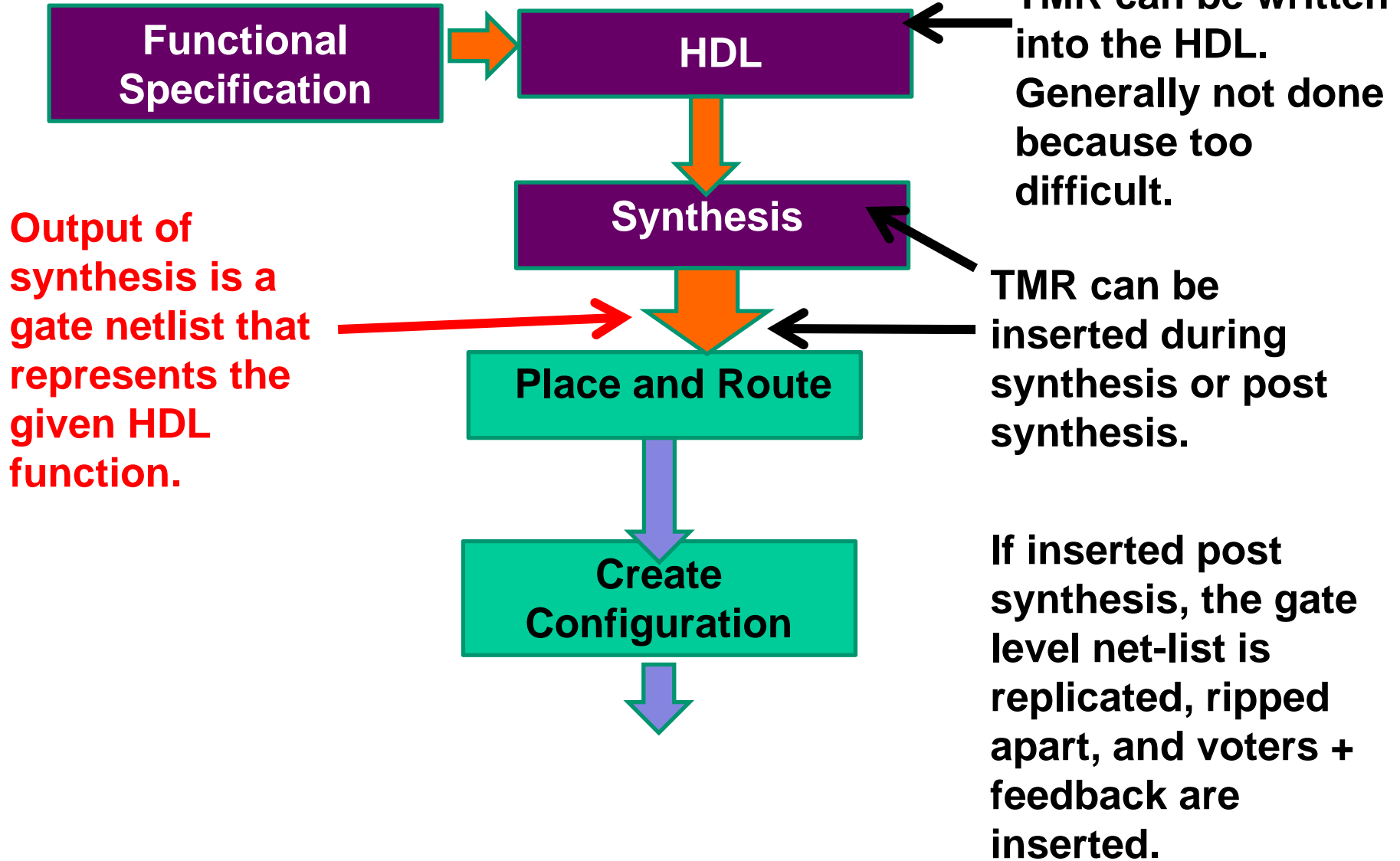
- **The previous slides were affiliated with system level mitigation.**
- **System level mitigation generally has:**
  - **Detection, masking, no correction, downtime, and recovery actions.**
- **The following slides will discuss triple modular redundancy (TMR) techniques that can be implemented as system or design-level mitigation.**
- **Most of the TMR techniques will incorporate masking and detection with no downtime (unless there is a single functional interrupt (SEFI)).**
- **Hence, TMR can improve system performance, availability, and elongate operation time.**



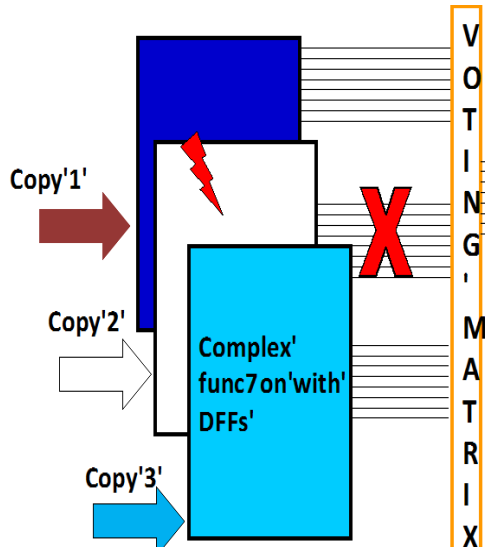
**Mitigation – Fail Safe Strategies That  
Do Not Require Fault Detection but  
Provide SEU Masking and/or  
Correction:  
Triple Modular Redundancy (TMR)...  
best two out of three.**

# How To Insert TMR into A Design:

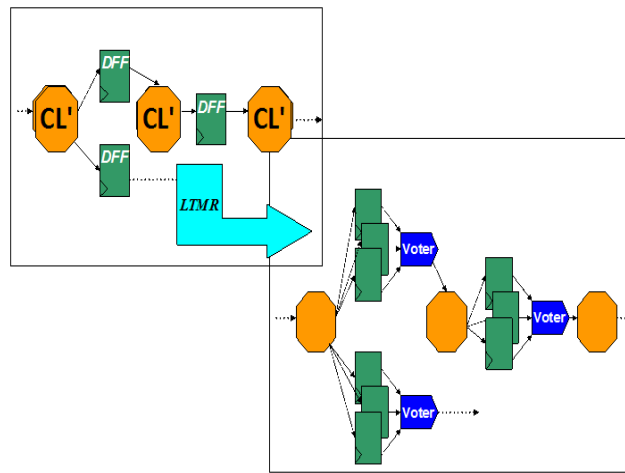
HDL: Hardware description language



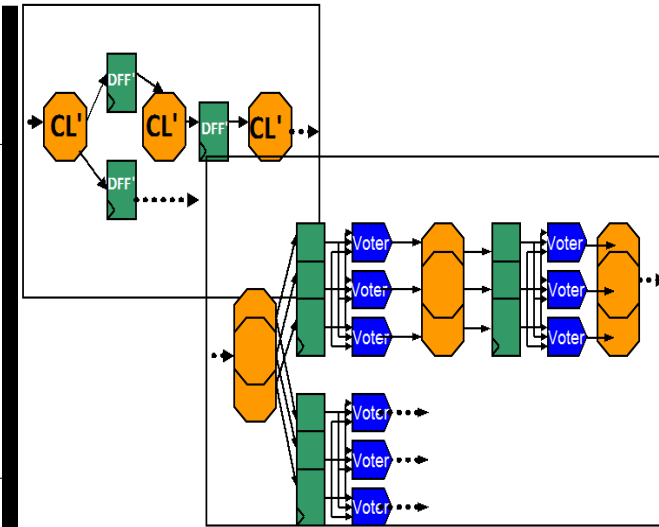
# Various TMR Schemes: Different Topologies



**Block diagram of block TMR (BTMR):** a complex function containing combinatorial logic (CL) and flip-flops (DFFs) is triplicated as three black boxes; majority voters are placed at the outputs of the triplet.



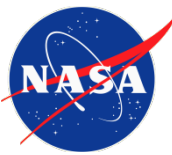
**Block diagram of local TMR (LTMR):** only flip-flops (DFFs) are triplicated and data-paths stay singular; voters are brought into the design and placed in front of the DFFs.



**Block Diagram of distributed TMR (DTMR):** the entire design is triplicated except for the global routes (e.g., clocks); voters are brought into the design and placed after the flip-flops (DFFs). DTMR masks and corrects most single event upsets (SEUs).

Same Definitions used by Mentor and Synopsys





# TMR Implementation

- As previously illustrated, TMR can be implemented in a variety of ways.
- The definition of TMR depends on what portion of the circuit is triplicated and where the voters are placed.
- The strongest TMR implementation will triplicate all data-paths and contain separate voters for each data-path.
  - However, this can be costly: area, power, and complexity.
  - Hence a trade is performed to determine the TMR scheme that requires the least amount of effort and circuitry that will meet project requirements.
- Presentation scope: Block TMR (BTMR), Localized TMR (LTMR), Distributed TMR (DTMR), Global TMR (GTMR).

# Block Triple Modular Redundancy: BTMR

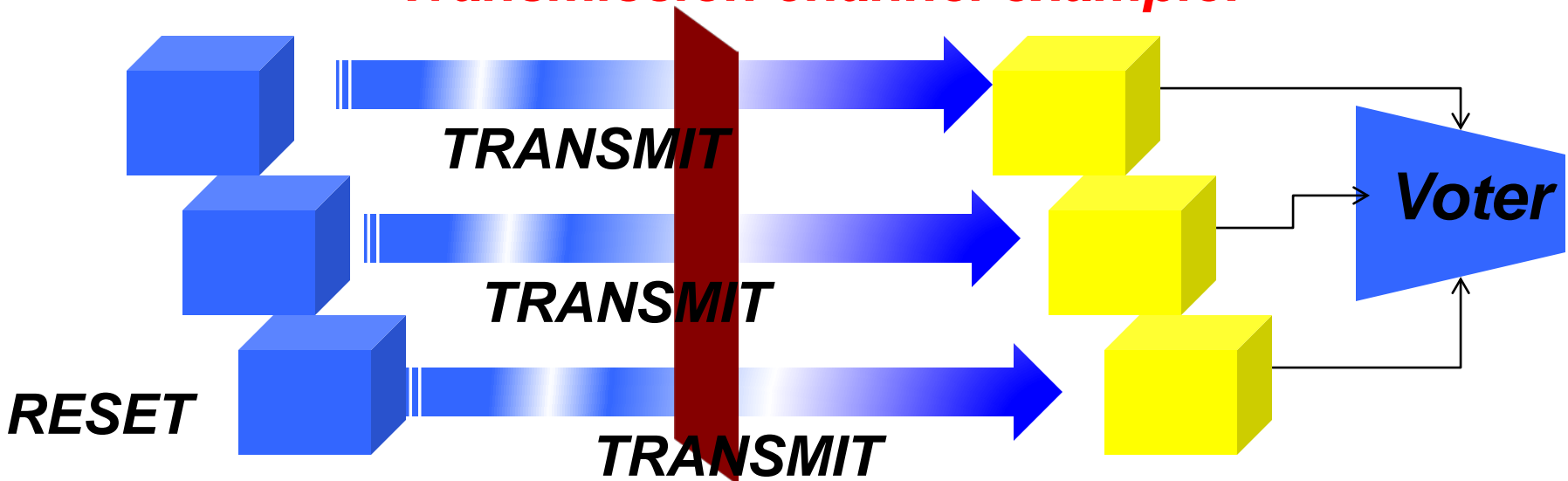


- **Need Feedback to Correct.**
- **Cannot apply internal correction from voted outputs.**
- **If blocks are not regularly flushed (e.g. reset), Errors can accumulate – may not be an effective technique.**

# Examples of a Flushable BTMR Designs

- Shift Registers.
- Transmission channels: It is typical for transmission channels to send and reset after every sent packet.
- Systems that can be reset (or power-cycled) every so-often... **Yes that includes processors.**

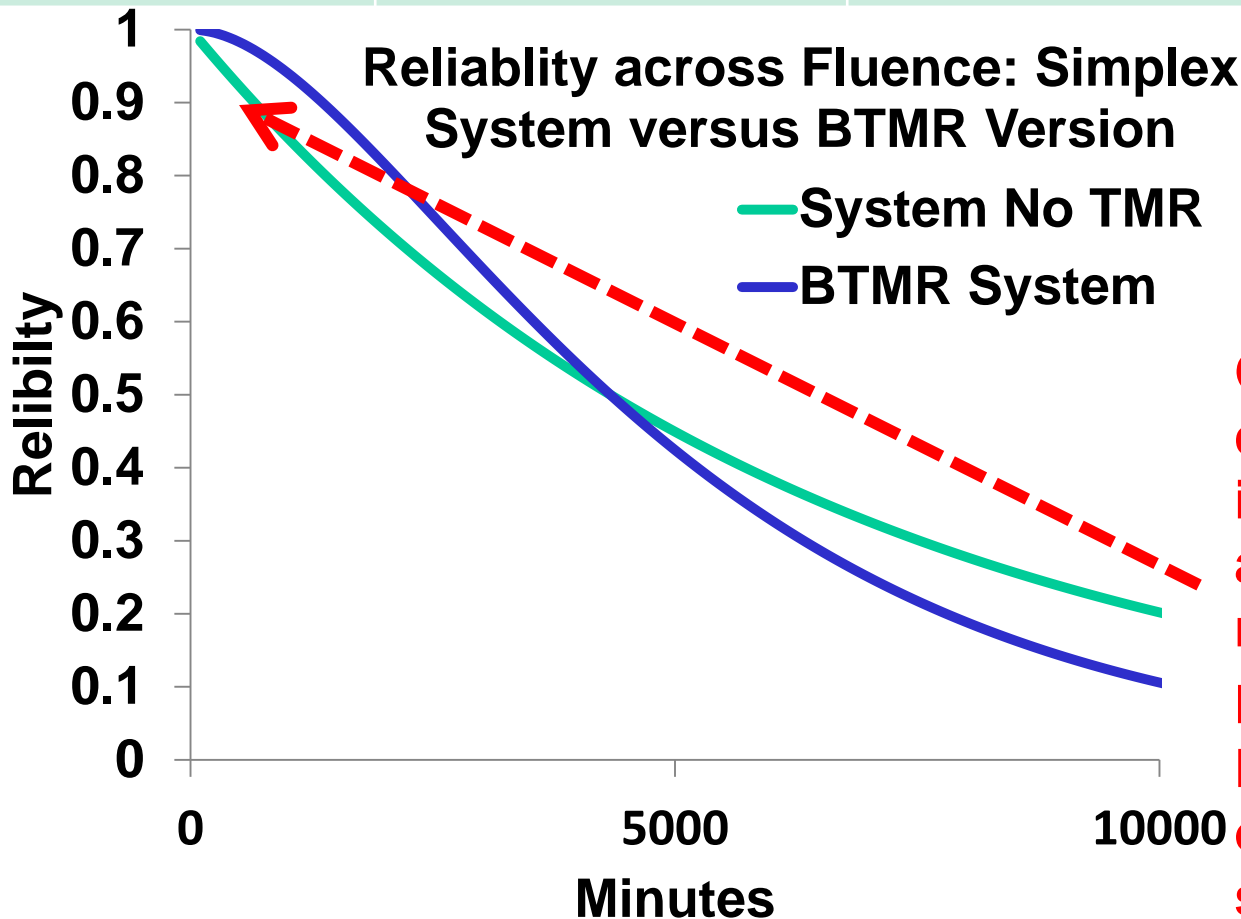
## *Transmission channel example:*



# Explanation of BTMR Strength and Weakness using Classical Reliability Models



Reliability for 1 block ( $R_{\text{block}}$ )	Reliability for BTMR ( $R_{\text{BTMR}}$ )	Mean Time to Failure for 1 block ( $\text{MTTF}_{\text{block}}$ )	Mean Time to Failure BTMR ( $\text{MTTF}_{\text{BTMR}}$ )
$e^{-\lambda t}$	$3 e^{-2\lambda t} - 2 e^{-3\lambda t}$	$1/\lambda$	$(5/6 \lambda) = 0.833/\lambda$



$$\lambda = \frac{\text{Failures}}{\text{Time}}$$

**Operating a BTMR design in this time interval will provide an increase in reliability.**

**However, over time, BTMR reliability drops off faster than a system with No TMR.**

# Classical Reliability: BTMR Bottom Line



- **Concerns and limitations:**
  - What is your reliable window of operation relative to the MTTF for one unmitigated block?
  - Overtime, a BTMR system has lower reliability than an unmitigated system.
  - Applying additional replicated blocks (e.g., N-out-of-M) will only increase the reliability during the short window near start time. However, overtime, the reliability of an N-out-of-M system will fall faster as M (the number of replicated blocks) grows.
- **Benefits!!!!**
  - BTMR can block an error from propagating to other areas of the system.
  - BTMR is a good (simple) solution for flushable-systems.



# Additional BTMR Warnings

- **With BTMR, not all I/O can be monitored.**
- **Should address first system failure when it occurs and correct system state.... And to do so... Usually need an additional detection signal to know when one of the systems are in failure.**
- **AVAILABILITY!**

# What Should be Done If Availability Needs to be Increased?



- If the blocks within the BTMR have a relatively high upset rate with respect to the availability window, then stronger mitigation must be implemented.
- Bring the voting/correcting inside of the modules... bring the voting to the module DFFs.

***The following slides illustrate the various forms of TMR that include voter insertion in the data-path.***

TMR Nomenclature	Description <i>DFF: Edge triggered flip-flop; CL: Combinatorial Logic</i>	TMR Acronym
Local TMR	DFFs are triplicated	LTMR
Distributed TMR	DFFs and CL-data-paths are triplicated	DTMR
Global TMR	DFFs, CL-data-paths and global routes are triplicated	GTMR or XTMR



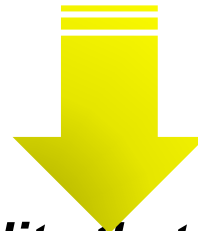
# Describing Mitigation Effectiveness Using A Model

*DFF: Edge triggered flip-flop*

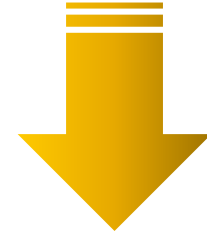
*CL: Combinatorial Logic*

$$P(fs)_{error} \propto P_{configuration} + P(fs)_{functionalLogic} + P_{SEFI}$$

$$P(fs)_{DFFSEU \rightarrow SEU} + P(fs)_{SET \rightarrow SEU}$$



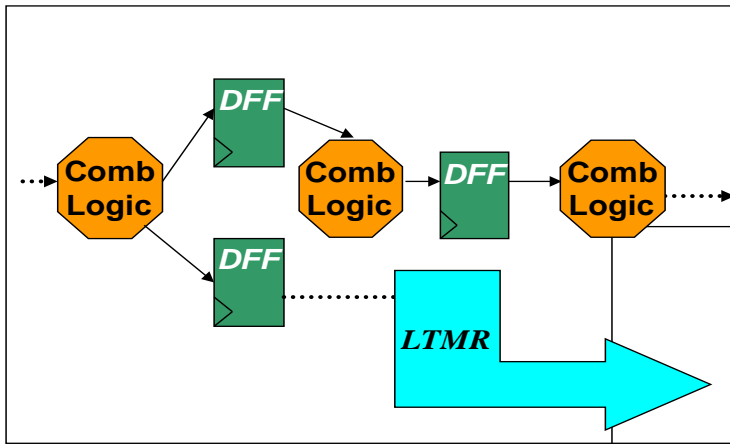
***Probability that an SEU in a DFF will manifest as an error in the next system clock cycle***



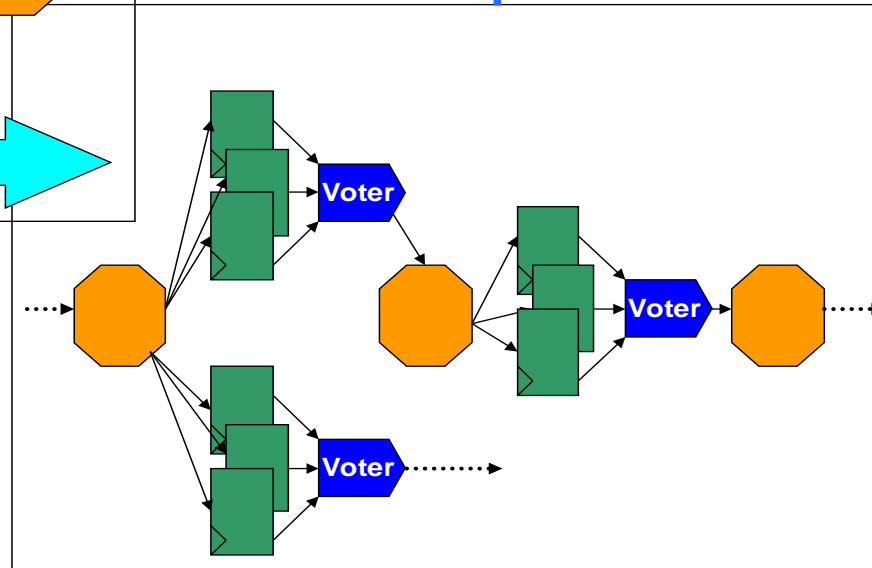
***Probability that an SET in a CL gate will manifest as an error in the next system clock cycle***

# Local Triple Modular Redundancy (LTMR)

- Only DFFs are triplicated. Data-paths are kept singular.
- LTMR masks upsets from DFFs and corrects DFF upsets if feedback is used.



- Good for devices where DFFs are most susceptible and configuration and CL susceptibility is insignificant; e.g., **Microsemi products.**

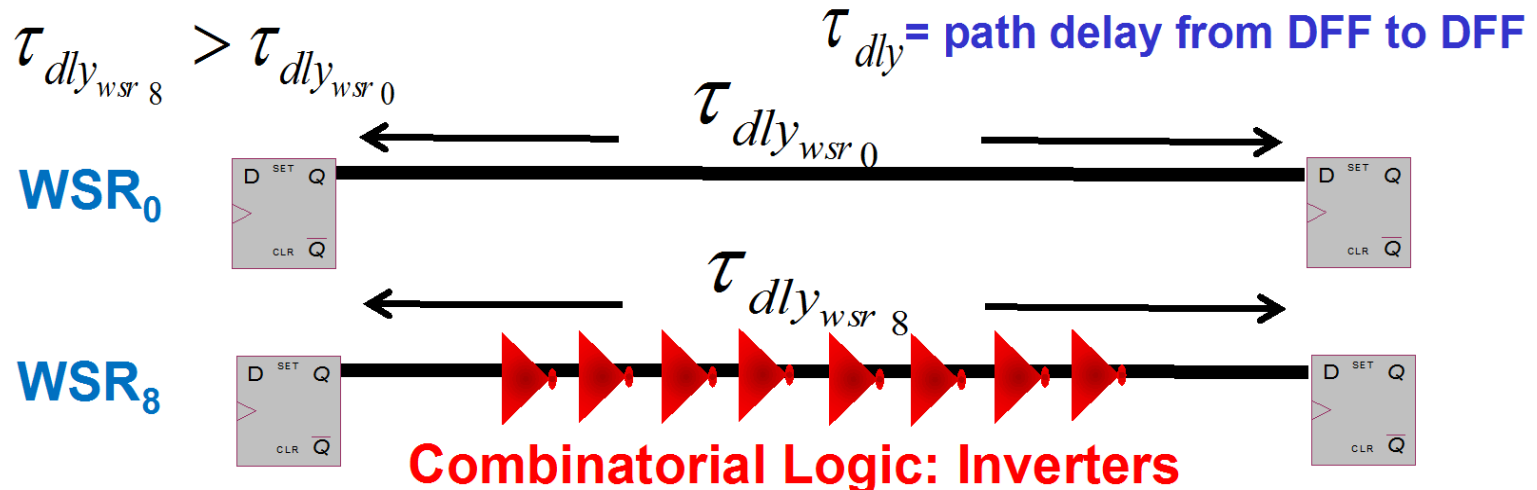
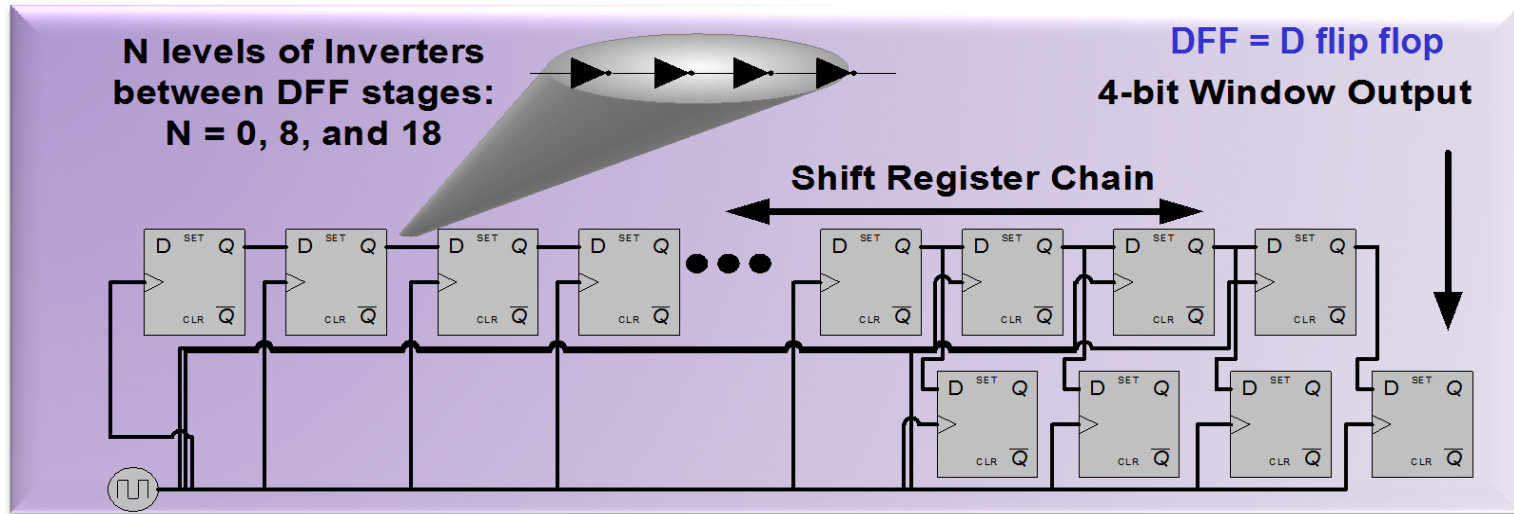


$$P(fs)_{error} \propto P_{configuration} + P(fs)_{functionalLogic} + P_{SEFI}$$

$$P(fs)_{DFF \rightarrow SEU} + P(fs)_{SET \rightarrow SEU}$$

A red arrow points from the first term of the equation above to a red '0', indicating that this term is zero.

# Windowed Shift Registers (WSRs): NEPP Test Structure



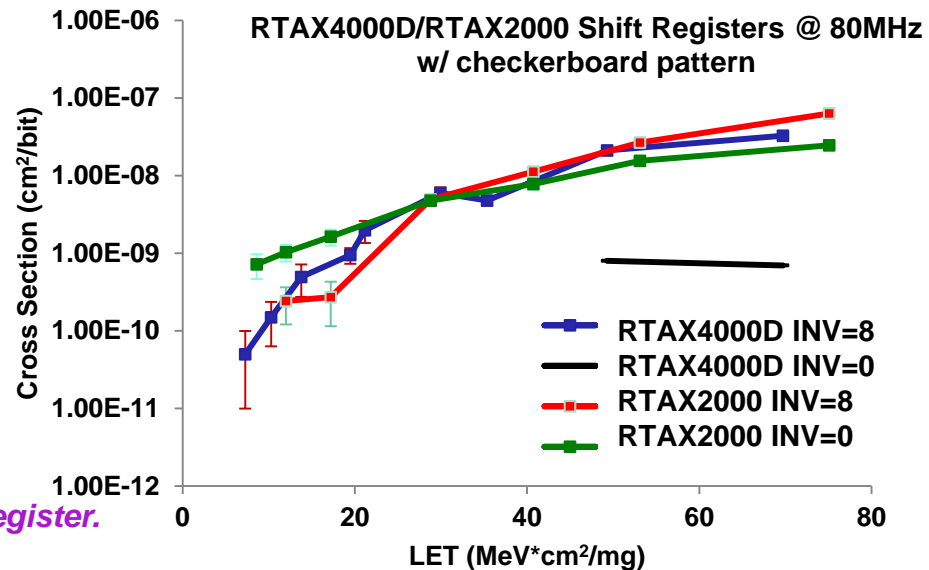
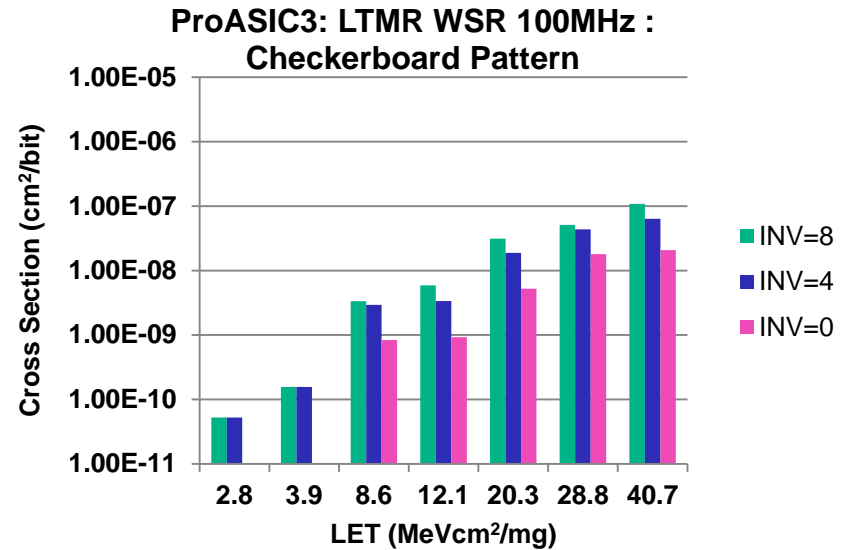


# Adding LTMR to a Microsemi ProASIC3 Device versus RTAXs Embedded LTMR

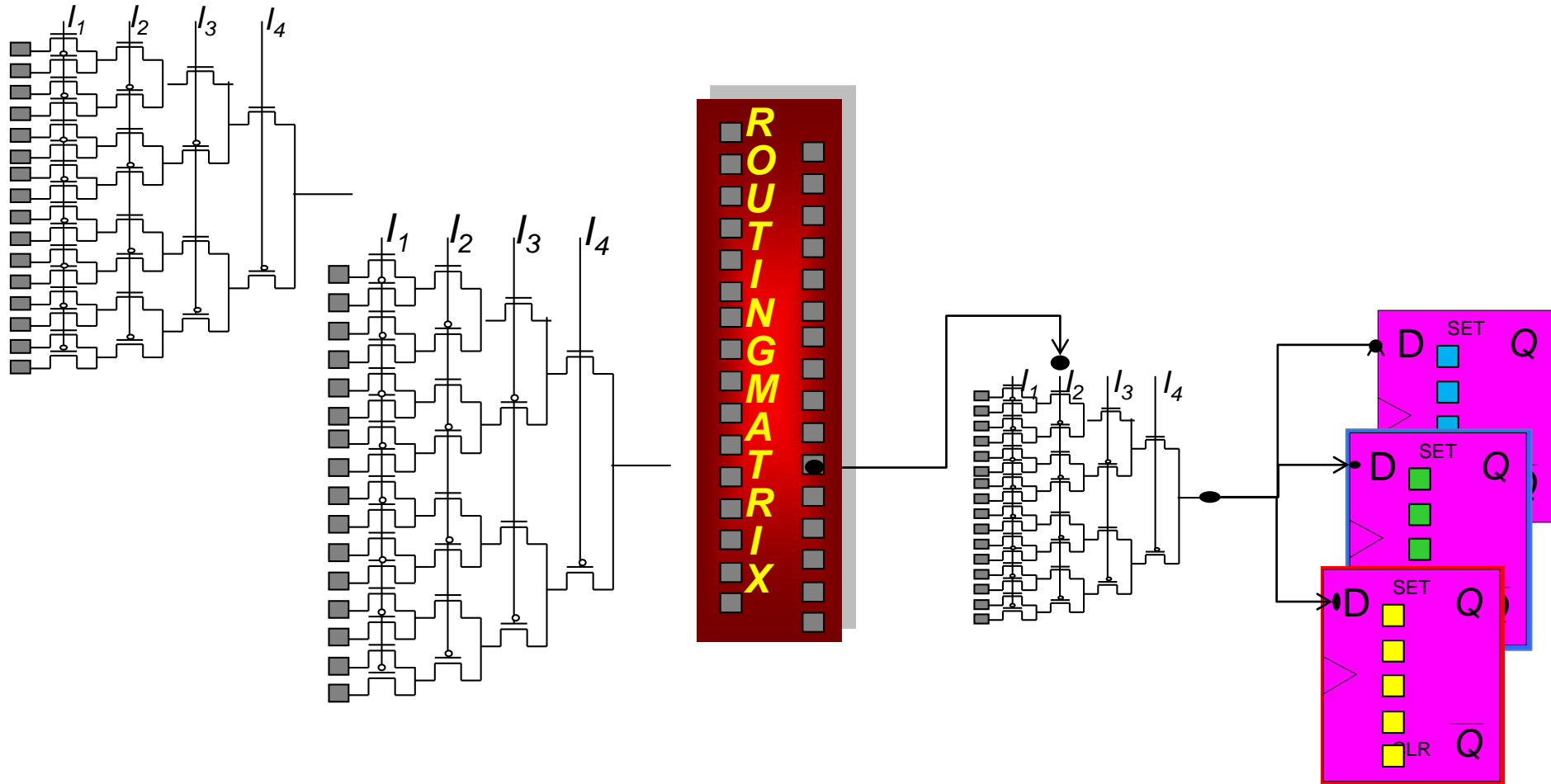
- At lower LETs, applying LTMR to a ProASIC3 design, has similar (a little higher) SEU response to Microsemi RTAXs series.
- At higher LETs, clock tree upsets start to dominate and LTMR in the ProASIC3 is not as effective.
- Depending on your target radiation environment, the ProASIC3 may be acceptable for your application.
- Note: RTAX2000 INV=8 has a lower SEU than RTAX2000 INV=0 at low LET. This is from SET filtering (high capacitive routing – 130nm antifuse).

*WSR: Test circuit...Windowed Shift Register.*

*INV: Inverters between WSR stages.*



# LTMR Should **Not** Be Used in An SRAM Based FPGA

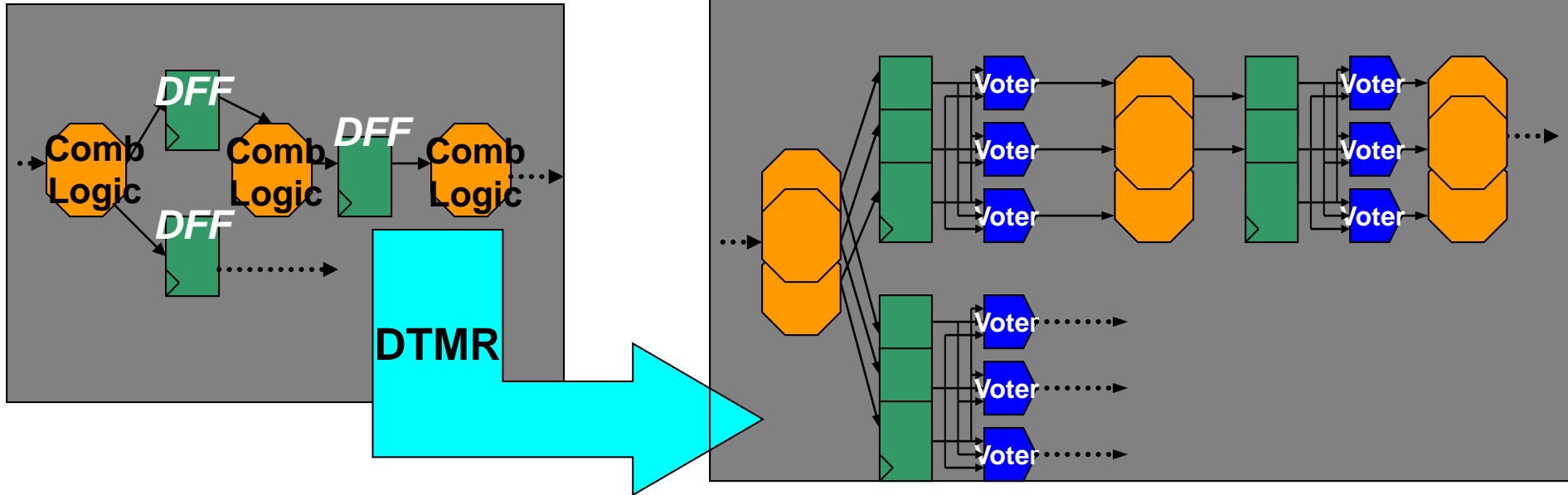


**Proven via NEPP experiments: SEU data for LTMR implemented in Xilinx FPGA devices are similar or worse than no added mitigation.**



# Distributed Triple Modular Redundancy (DTMR)

- Triple all data-paths and add voters after DFFs (**NOT Clocks**) .
- DTMR masks upsets from configuration + DFFs + CL and corrects captured upsets if feedback is used.
- Good for devices where configuration or DFFs + CL are more susceptible than project requirements; e.g., **Xilinx and Altera commercial FPGAs.**



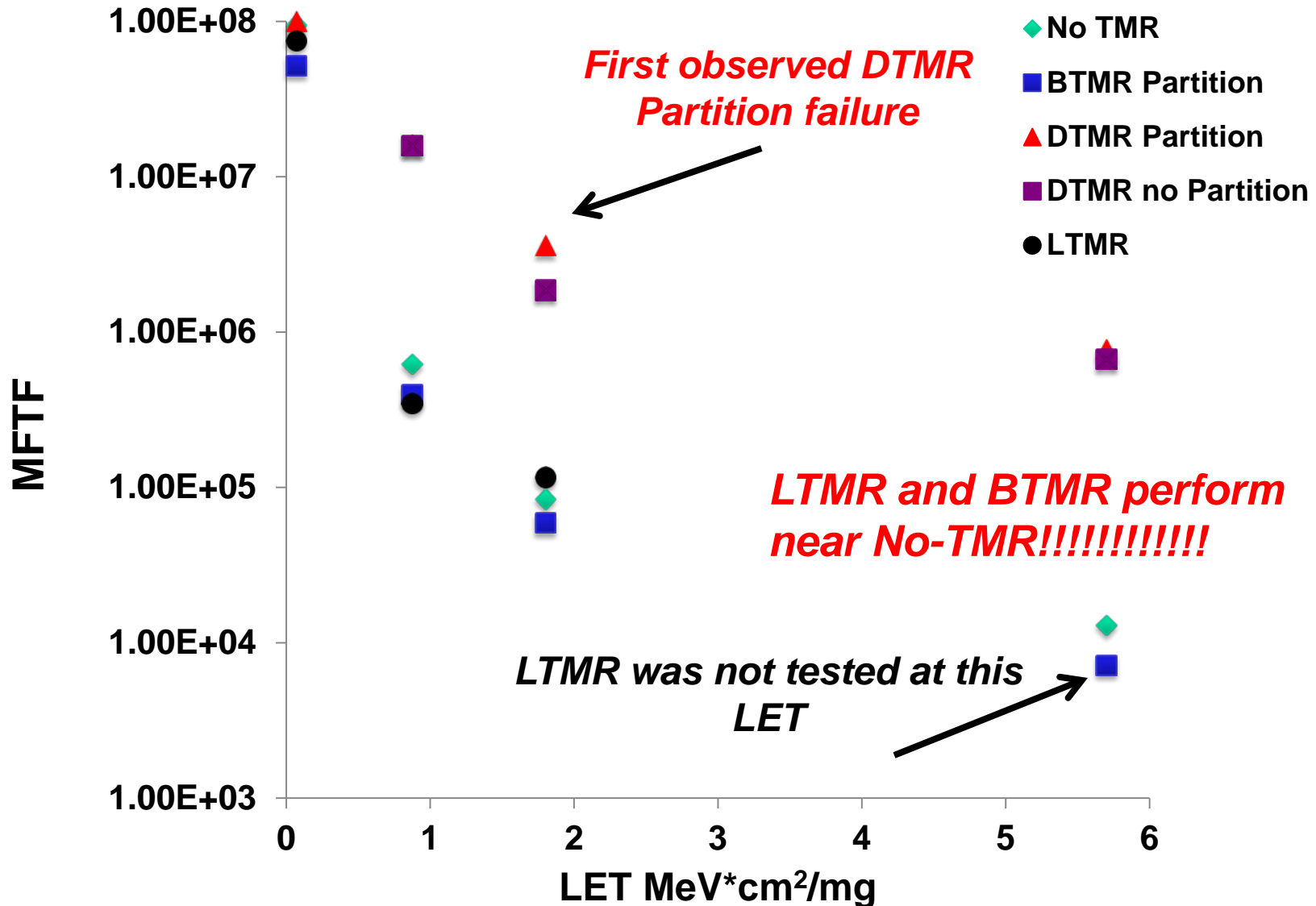
$$P(f_s)_{error} \propto P_{configuration} + P(f_s)_{functionalLogic} + P_{SEFI}$$

Low Minimally Lowered

$$P(f_s)_{DFFSEU \rightarrow SEU} + P(f_s)_{SET \rightarrow SEU}$$

Low

# Xilinx Kintex UltraScale Mitigation Study: 8-bit Counters



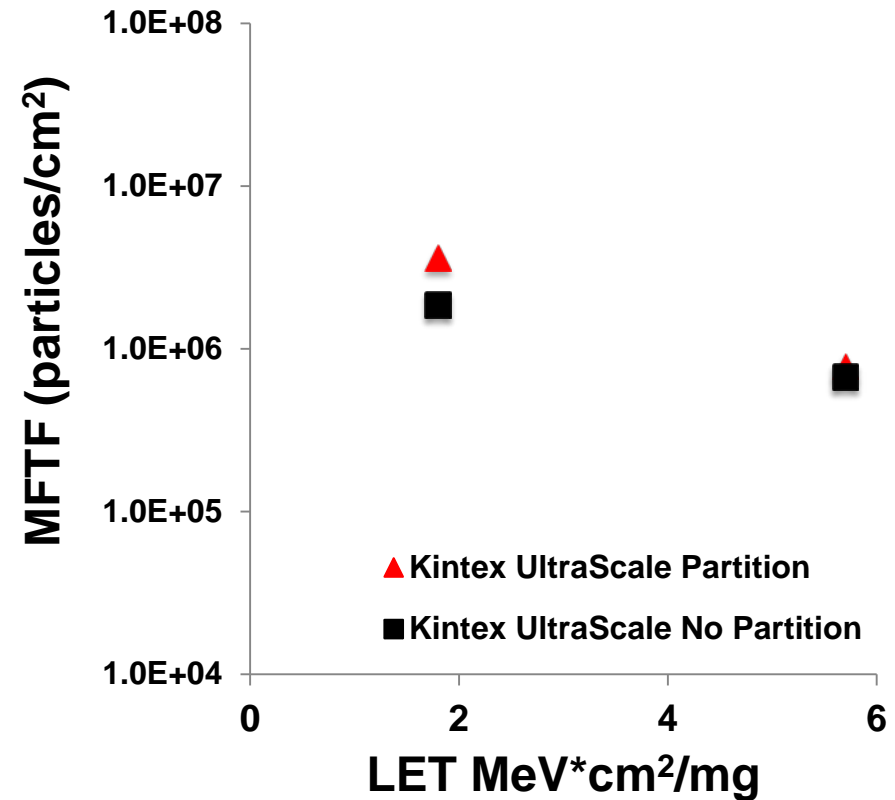
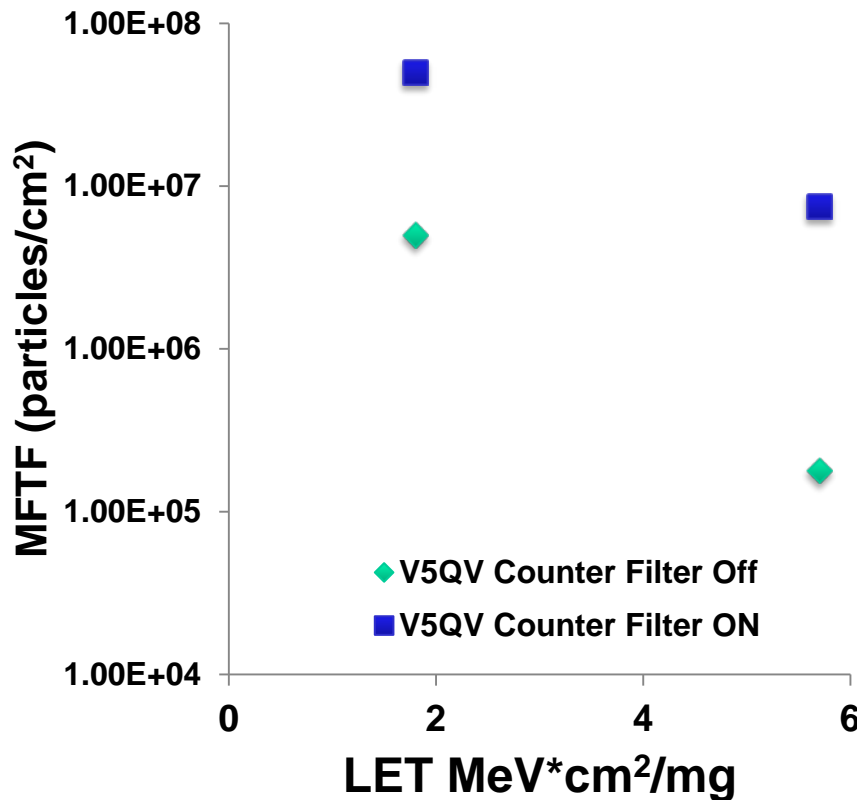


# Comparison of V5QV and Kintex-UltraScale with Mitigation



**V5QV Counters:  
Embedded Mitigation**

**Kintex UltraScale DTMR Counters:  
User Inserted Mitigation**



***DTMR inserted with Synopsys synthesis tool***

# Theoretically, GTMR Is The Strongest Mitigation Strategy... BUT...



- Triplicate all clocks, data-paths and add voters after DFFs
- Triplicating a design and its global routes takes up a lot of power and area.
- Skew between clock domains must be minimized such that it is less than the shortest routing delay from DFF to DFF (hold time violation or race condition):
  - Is skew between clock trees in the FPGA small enough? **Most likely not.**
  - Limit skew of clocks coming into the FPGA.
  - Limit skew of clocks from their input pin to their clock tree.
- Difficult to verify.



# TMR and Verification

- If a system is required to be protected using TMR, improper insertion can jeopardize the reliability and security of the system.
- There are two primary concerns to TMR insertion:
  - Did the insertion cause incorrect logic implementation?
  - Is the insertion the correct topology?
    - Are all voters inserted where expected? and
    - Are all components triplicated as expected?
  - Example: must be able to differentiate between LTMR, DTMR, or some other implementation.
- Due to the complexity of the verification process and the complexity of digital designs, there are currently no available techniques that can provide complete and reliable confirmation of TMR insertion.
- Critical applications: if protection is required, then its implementation must be assured.

***We are working on it!***



# TMR Rules of Thumb

- **FPGAs with embedded mitigation do not usually require additional (user inserted) TMR.**
- **FPGAs with soft configuration will only benefit from DTMR or BTMR (in appropriate situations).**
- **FPGAs with hard configuration and no other embedded mitigation will benefit from local mitigation strategies.**

# TMR Warnings

- **There are significant differences between TMR schemes. Select the correct type for your application and requirements.**
- **Do not use LTMR in a Xilinx Device.**
- **BTMR is a sufficient mitigation strategy if the required reliability window is relatively small as compared to MTTF of a non-redundant (non-mitigated) system.**
- **Most FPGAs cannot accommodate the clock skew between clock trees to properly implement GTMR. Best to stay away.**
- **TMR is difficult to verify. Fault injection is not sufficient for critical applications.**



# Some Thoughts

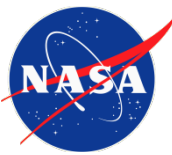


# Concerns and Challenges of Today and Tomorrow for Mitigation Insertion (1)



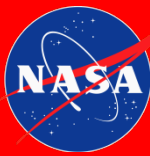
- **User insertion of mitigation strategies in most FPGA and ASIC devices has proven to be a challenging task because of reliability, performance, area, and power constraints.**
  - **Difficult to synchronize across triplicated systems,**
  - **Mitigation insertion slows down the system.**
  - **Can't fit a triplicated version of a design into one device.**
  - **Power and thermal hot-spots are increased.**
- **The newer commercial devices have a significant increase in gate count and lower power. This helps to accommodate for area and power constraints while triplicating a design. However, this increases the challenge of module synchronization.**





# Concerns and Challenges of Today and Tomorrow for Mitigation Insertion (2)

- **Embedded mitigation has helped in the design process. However, it is proving to be an ever-increasing challenge for manufacturers.**
  - We (users) want embedded mitigation: cheaper design flow process, faster, and less power hungry.
  - However, heritage has proven that for critical applications, embedded systems have provided excellent performance and reliability.
- **Tool availability... Getting better... IP Cores are still problematic.**
- **User's are not selecting the correct mitigation scheme for their target FPGA.**
- **Mitigation is too complex to fully verify.**



# Warning

- **You should not mitigate failure mechanisms that have insignificant contribution to the overall failure rate:**
  - This adds risk.
  - Slows down system.
  - Can provide a false sense of protection.
  - Gain is not significant.

$$P(fS)_{error} \propto P(fS)_{Configuration} + P(fS)_{functionalLogic} + P(fS)_{SEFI}$$



# Summary

- For critical applications, mitigation might be required.
- Determine the correct mitigation scheme for your mission while incorporating given requirements:
  - Understand the susceptibility of the target FPGA and potential necessity of other devices.
  - Investigate if the selected mitigation strategy is compatible to the target FPGA device.
  - Calculate the reliability of the mitigation strategy to determine if the final system will satisfy requirements.
  - **Ask the right questions regarding functional expectation, mitigation, requirement satisfaction, and verification of expectations.**
- Although it is desirable from a user's perspective to have embedded mitigation, cost seems to be driving the market towards unmitigated commercial FPGA devices. Hence, it will be necessary for user's to familiarize themselves with optimal mitigation insertion and usage.