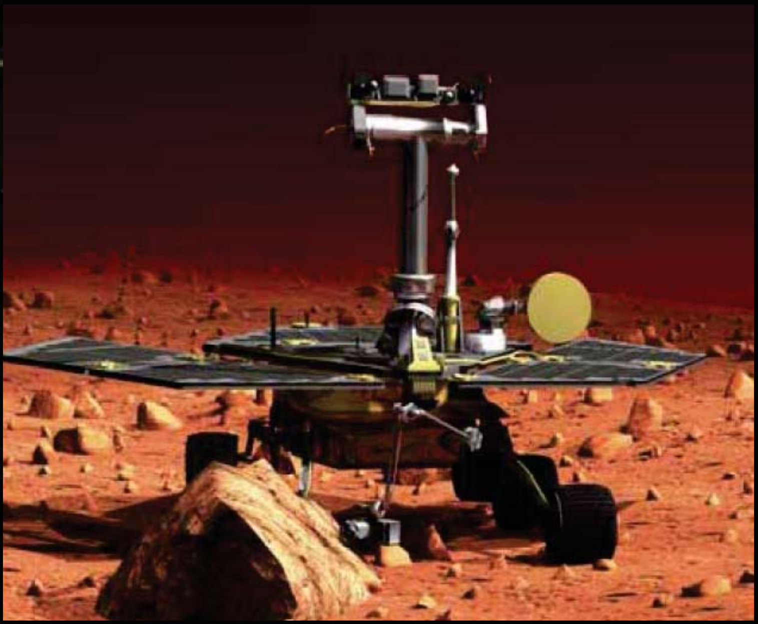
New Developments in FPGA: SEUs and Fail-Safe Strategies from the NASA Goddard Perspective



Melanie Berg, AS&D in support of NASA/GSFC
Melanie.D.Berg@NASA.gov
Kenneth LaBel, NASA/GSFC
Jonathan Pellish, NASA/GSFC

To be presented by Melanie Berg via WebEx at SERRESSA 2015 International School on the Effects of Radiation on Embedded Systems for Space Applications, Puebla, Mexico, November 30 to December 4, 2015.

Acknowledgements

- ***Some of this work has been sponsored by the NASA Electronic Parts and Packaging (NEPP) Program and the Defense Threat Reduction Agency (DTRA).***
- ***Thanks is given to the NASA Goddard Radiation Effects and Analysis Group (REAG) for their technical assistance and support. REAG is led by Kenneth LaBel and Jonathan Pellish.***

Contact Information:

Melanie Berg: NASA Goddard REAG FPGA Principal Investigator:

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)
- Combinatorial logic (CL)
- Configurable Logic Block (CLB)
- Digital Signal Processing Block (DSP)
- Distributed triple modular redundancy (DTMR)
- Edge-triggered flip-flops (DFFs)
- Equivalence Checking (EC)
- Error detection and correction (EDAC)
- Field programmable gate array (FPGA)
- Finite State Machine (FSM)
- Gate Level Netlist (EDF, EDIF, GLN)
- Global triple modular redundancy (GTMR)
- Hardware Description Language (HDL)
- Input – output (I/O)
- Linear energy transfer (LET)
- Local triple modular redundancy (LTMR)
- Look up table (LUT)
- Multiple Bit Upsets (MBUs)
- Naval Research Laboratory (NRL)
- Operational frequency (*fs)*
- Power on reset (POR)
- Place and Route (PR)
- Radiation Effects and Analysis Group (REAG)
- Single error correction double error detection (SECDED)
- 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 (σ_{SEU})
- Static random access memory (SRAM)
- System on a chip (SOC)

Agenda

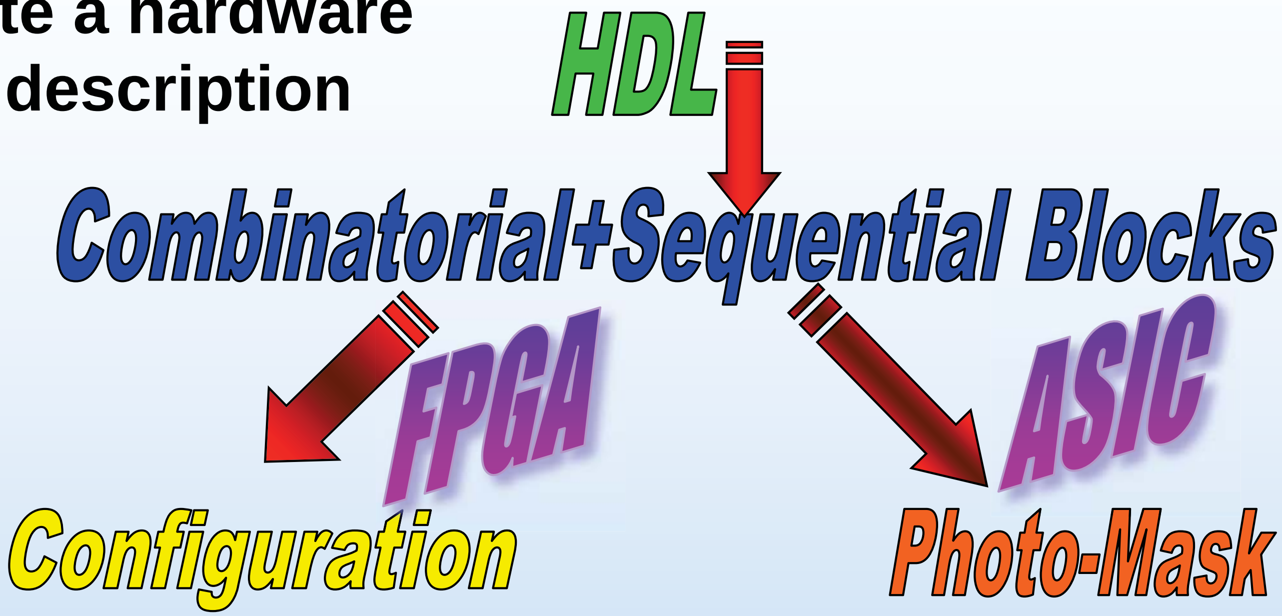
- **Single Event Upsets (SEUs) in FPGAs and Fail-Safe Overview.**
- **Single Event Upsets and FPGA Configuration.**
- **Single Event Upsets in an FPGA's Functional Data Path and Fail-Safe Strategies.**
- **Fail-Safe Strategies for FPGA Critical Applications.**
- **Fail-Safe State Machines.**

Definitions

- **A Field-Programmable Gate Array (FPGA) is a semiconductor device containing configurable logic components called "logic blocks", and configurable interconnects. Logic blocks can be configured to perform the function of basic logic gates such as AND, and XOR, or more complex combinational functions such as decoders or mathematical functions.**
- **An application-specific integrated circuit (ASIC) is a semiconductor device designed for a particular use. Its designs are considered more custom. Processors, RAM, ROM, etc… are examples of ASICs.**
- ***From a user's perspective, an FPGA is an ASIC designed to have a "sea" of configurable logic for general purpose usage.***

Creating A Design in An Integrated Circuit Device (FPGA or ASIC)

- **The objective is to create a hardware design using hardware description language (HDL):**
  - **Clocks,**
  - **Resets,**
  - **Sequential elements (e.g., flip-flops),**
  - **Combinatorial logic.**



HDL

Combinatorial+Sequential Blocks

FPGA

ASIC

Configuration

Photo-Mask

- **The description gets synthesized into a hardware gate-level-netlist (GLN: file listing gates and connectivity).**
- **The synthesized hardware gates are mapped and placed into the cell library (or logic blocks) of the target FPGA or ASIC.**
- **ASIC flow produces a mask that is handed to a foundry. FPGA flow produces a configuration file.**

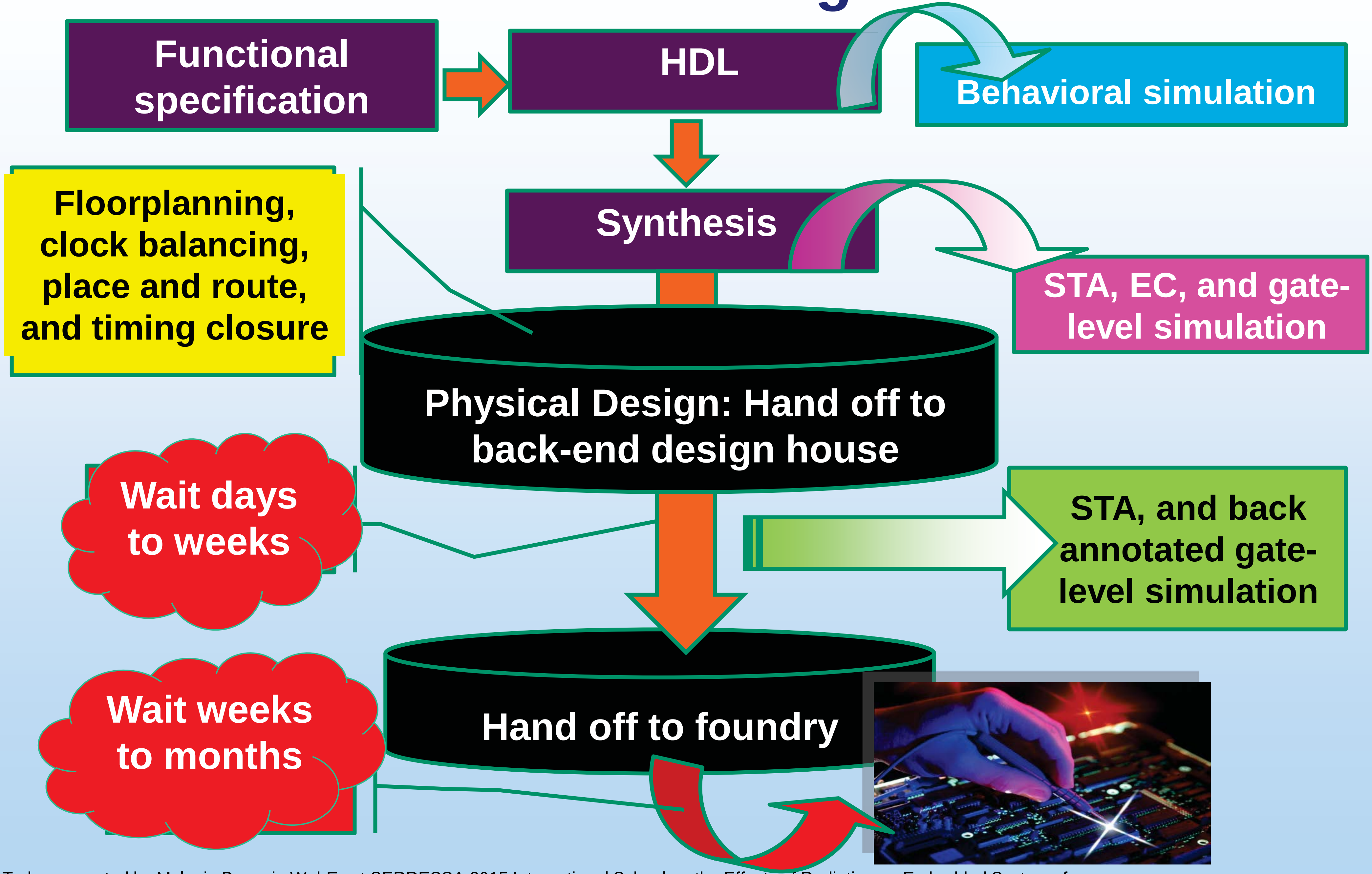
Design Tools

- **Design tools are used for each step of the design process.**
- **Synthesis: maps HDL into logic blocks (cells) … outputs gate-level net-lists.**
- **Place and route (PR): optimizes where the logic blocks and their interconnects should be within the device.**
- **Synthesis along with PR tools contain optimization algorithms within their tool sets.**
  - **These algorithms are used to optimize area, power, and logic function.**
  - **Tools are difficult to create. Poorly designed tools can create designs that are: functionally incorrect, too large to fit into the target device, or output too much power. Hence, a bad tool can produce unusable designs.**
  - **Equivalence checking (EC) verifies tool output matches HDL.**

Best practice is to use a proven vendor’s tool set – or product might be unreliable or unusable.

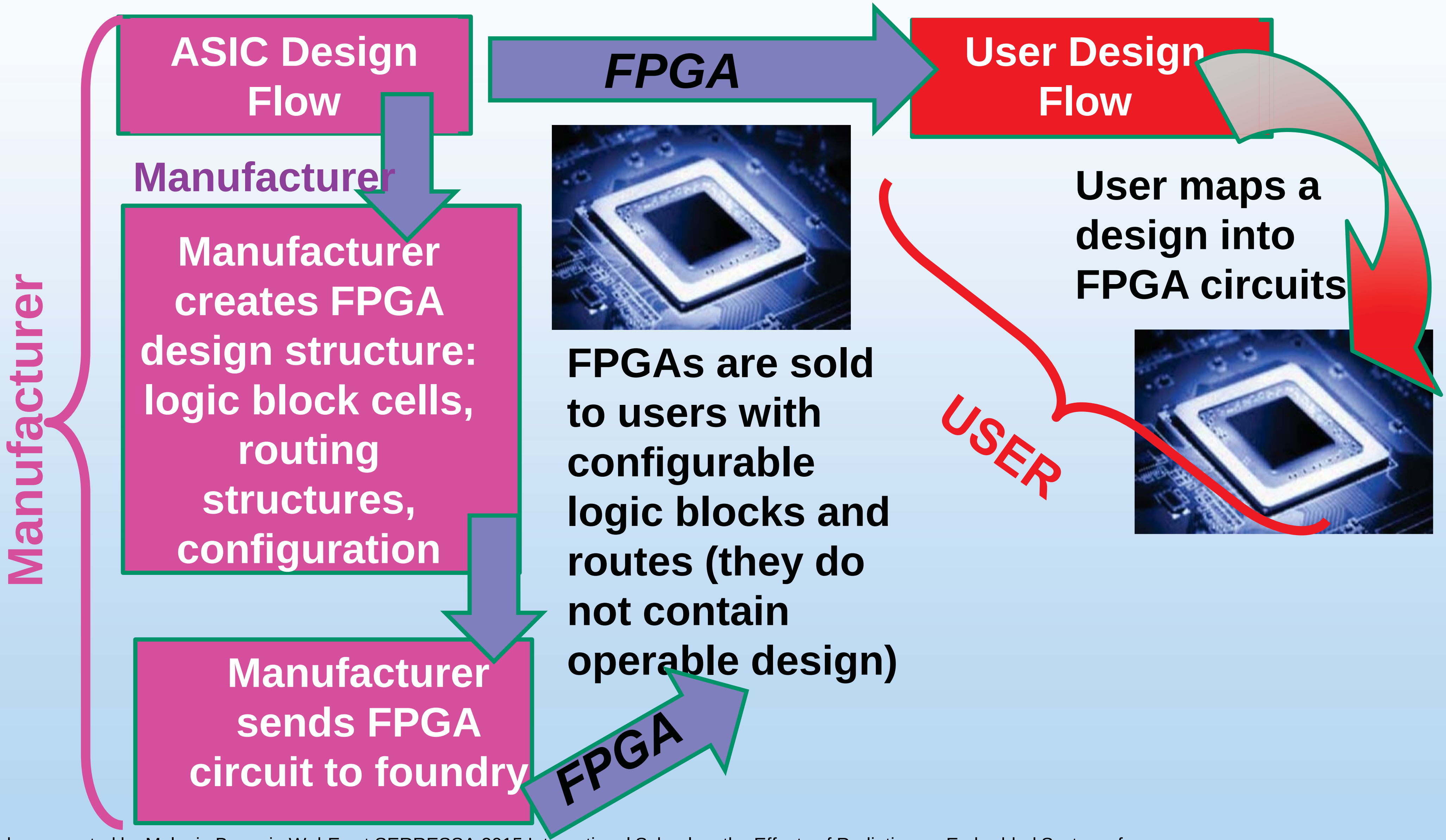
HDL: Hardware description language
STA: Static timing analysis
EC: Equivalence checking

ASIC Design Flow



FPGA Design Flow

FPGAs are created by manufacturers and are sold to users. The user maps a design into the FPGA fabric.

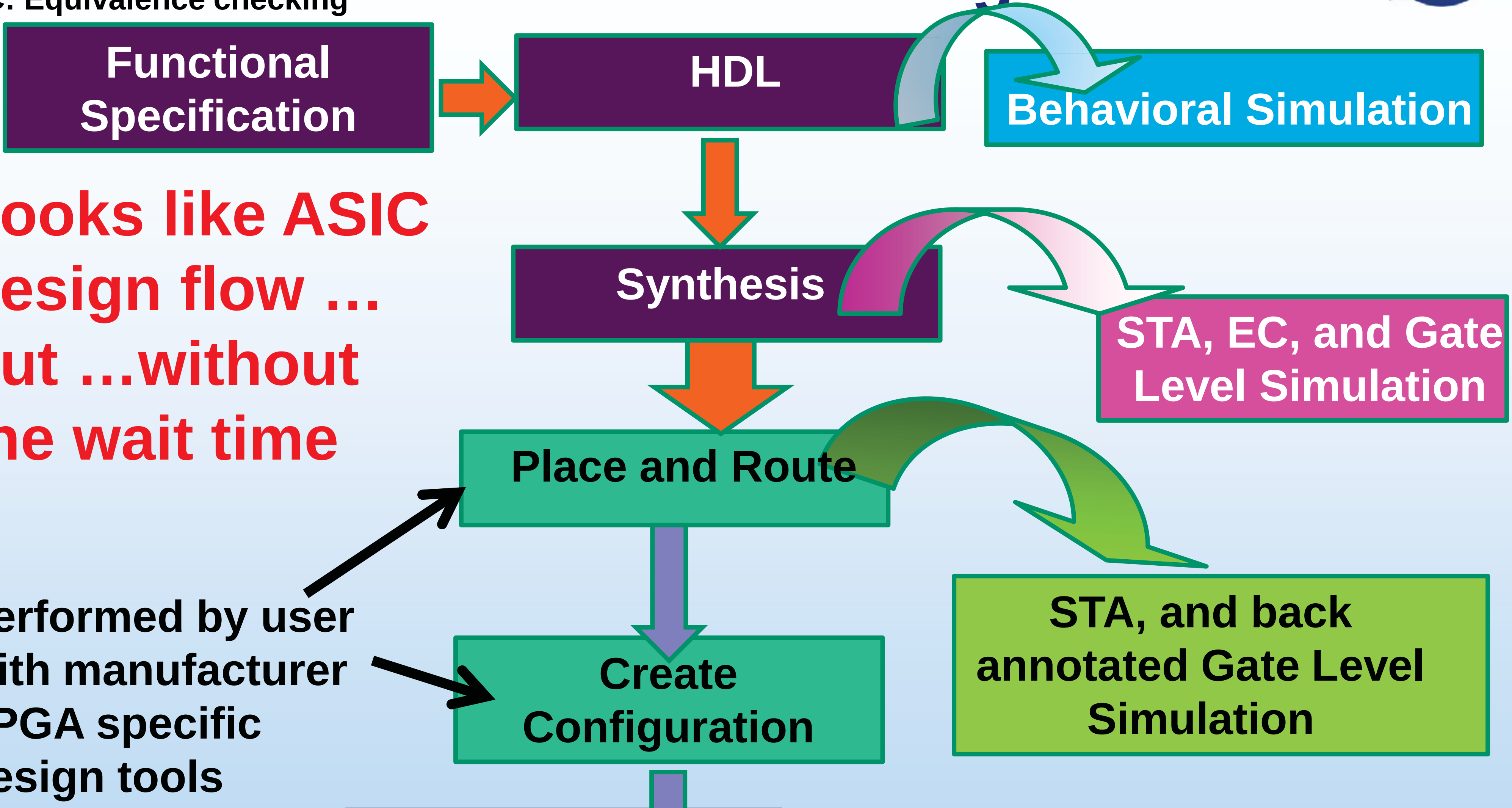


FPGAs are sold to users with configurable logic blocks and routes (they do not contain operable design)

User maps a design into FPGA circuits

HDL: Hardware description language
STA: Static timing analysis
EC: Equivalence checking

FPGA User Design Flow


Functional Specification

HDL

Behavioral Simulation

Synthesis

STA, EC, and Gate Level Simulation

Place and Route

STA, and back annotated Gate Level Simulation

Create Configuration

Looks like ASIC design flow … but …without the wait time

Performed by user with manufacturer FPGA specific design tools

User creates a design that is mapped into a manufacturer provided FPGA

FPGA or ASIC?

FPGA and ASIC Devices … System Usage

- An FPGA (similarly to an ASIC) can be used to solve any problem which is computable:
  - User implements a digital (or mixed signal design).
  - Design can be trivial glue-logic (e.g., interface control) or
  - Design can be as complex as a system on a chip that may include processors, embedded memory, and high speed serial interfaces (Gigabit SERDES). *SERDES: serializer de-serializer*
- The number of gates contained within the original FPGA devices were too small to compete with the ASIC devices of that time (1980s).
  - FPGAs were mostly used as interface glue logic.
  - Reduced system cost and added flexibility.
- Modern-day FPGAs contain millions of gates and have taken over a significant amount of the ASIC market.

The ASIC Advantage

ASIC Advantage	Comment/Explanation
Full custom capability	The design is “tailored” and is manufactured to design specifications (no additional hidden logic)
Lower unit costs	Great for very high volume projects
Smaller form factor	Less logic is required because device is manufactured to design specs
No configuration	Overall reliability can decrease due to the addition of configuration technology/logic
Lower power	Less logic is required because device is manufactured to design specs

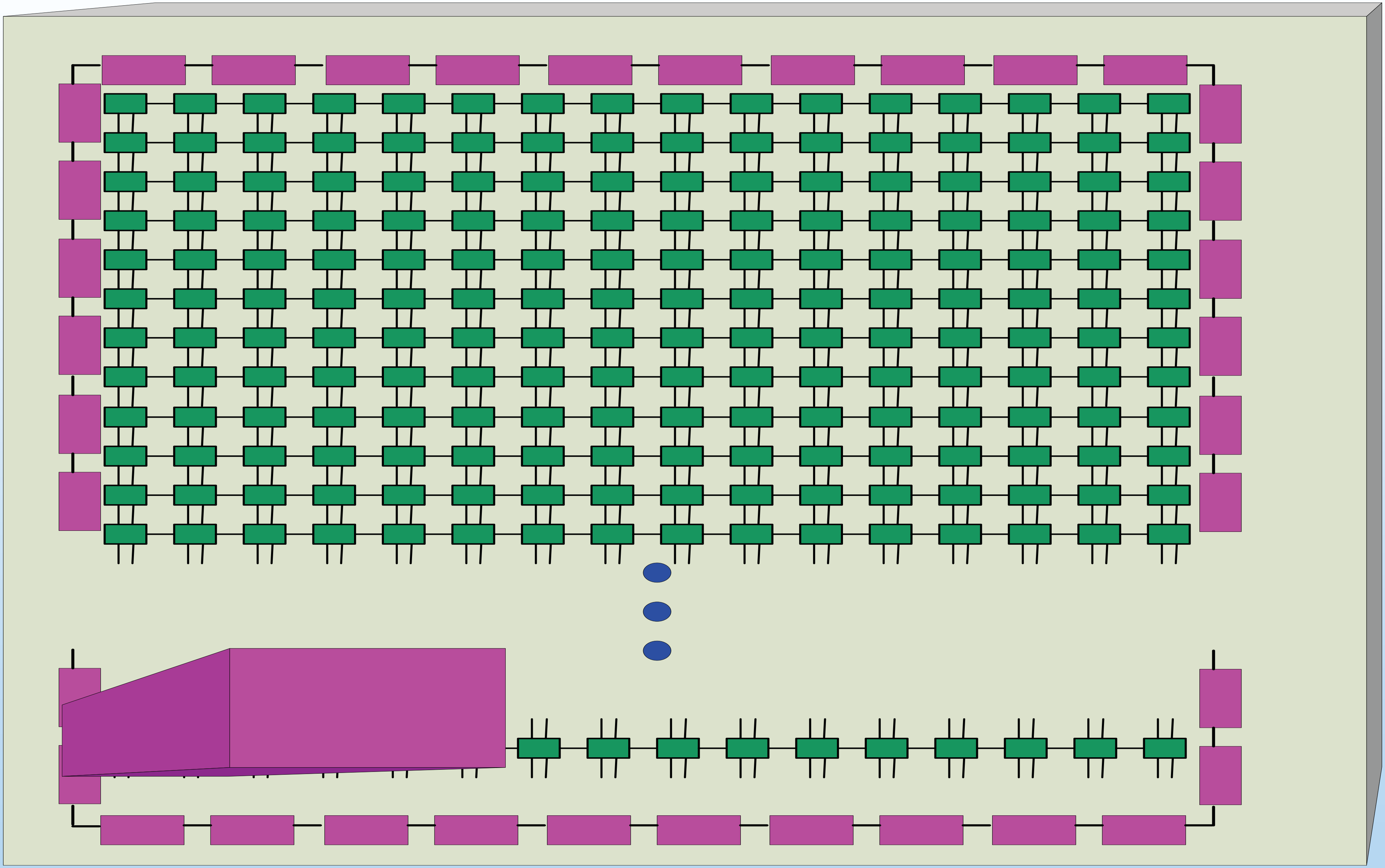
The FPGA Advantage

FPGA Advantage	Comment/Explanation
Faster time-to-market	No layout, masks or other manufacturing steps are needed
No upfront non-recurring expenses (NRE)	Costs typically associated with an ASIC design
Simpler design cycle	Due to the required tools that handle routing, placement, and timing
More predictable project cycle	Due to elimination of potential re-spins and lack of concern regarding wafer capacities as it would be in ASICs
Field reprogramability	It is easier to change a design in a system
Engineer availability	More students are taught FPGA design in school

FPGA: Faster design cycle and cheaper to implement

What’s Inside FPGA Devices?

General FPGA Architecture: Fabric Containing Customizable Preexisting Logic…User Building Blocks



To be presented by Melanie Berg via WebEx at SERRESSA 2015 International School on the Effects of Radiation on Embedded Systems for Space Applications, Puebla, Mexico, November 30 to December 4, 2015.

FPGA Structure Categorization as Defined by NASA Goddard REAG:

Single event functional interrupts (SEFI)
SEFI out of presentation scope

SEU cross section: σ_{SEU}

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

Design σ_{SEU} Configuration σ_{SEU} Functional logic σ_{SEU} SEFI σ_{SEU}

Sequential and Combinatorial logic (CL) in data path

Global Routes and Hidden Logic

SEU Testing is required in order to characterize the σ_{SEU}s for each of FPGA categories.

How Do FPGA's Differ?

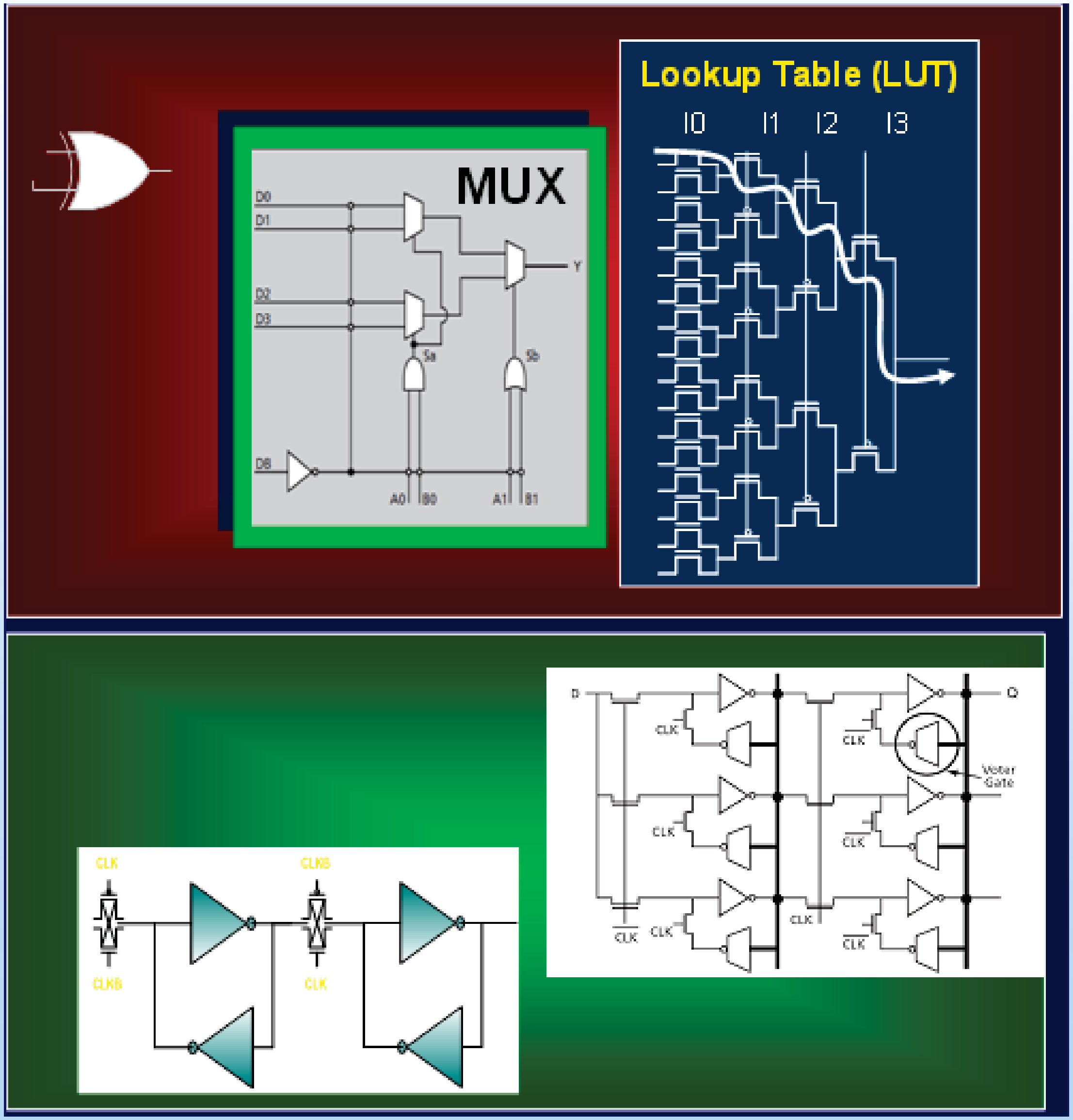
- **Manufacturer Architecture (not all are listed):**
  - **Configuration,**
  - **User building blocks (combinatorial logic cells, sequential logic cells),**
  - **Routing,**
  - **Clock structures,**
  - **Embedded mitigation, and**
  - **Embedded intellectual property (IP); e.g., memories, complex I/O management, phase locked loops (PLLs), and processors.**
- **Manufacturer design tool environment:**
  - **Synthesis,**
  - **Place and Route, and**
  - **Configuration management output.**

Difference in architectures and tools will affect the final design and design process – users be aware.

FPGA Component Libraries: Basic Designer Building Blocks (They Differ per FPGA Type)

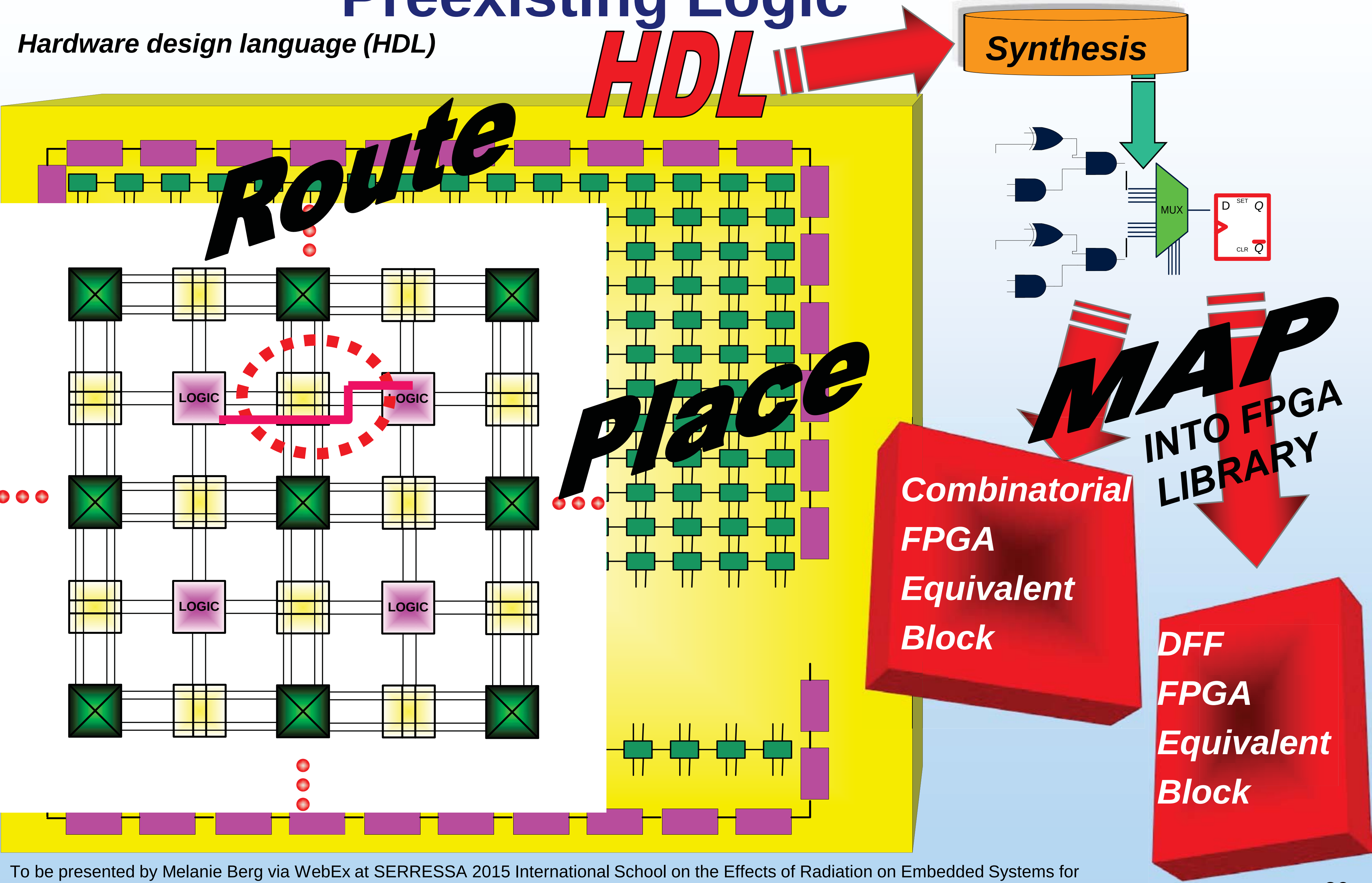
- **Combinatorial logic (CL) blocks**
  - **Vary in complexity.**
  - **Vary in I/O.**

- **Sequential logic blocks (DFF)**
  - **Uses global Clocks.**
  - **Uses global Resets.**
  - **May have mitigation.**



User Maps the Design Logic into FPGA Preexisting Logic

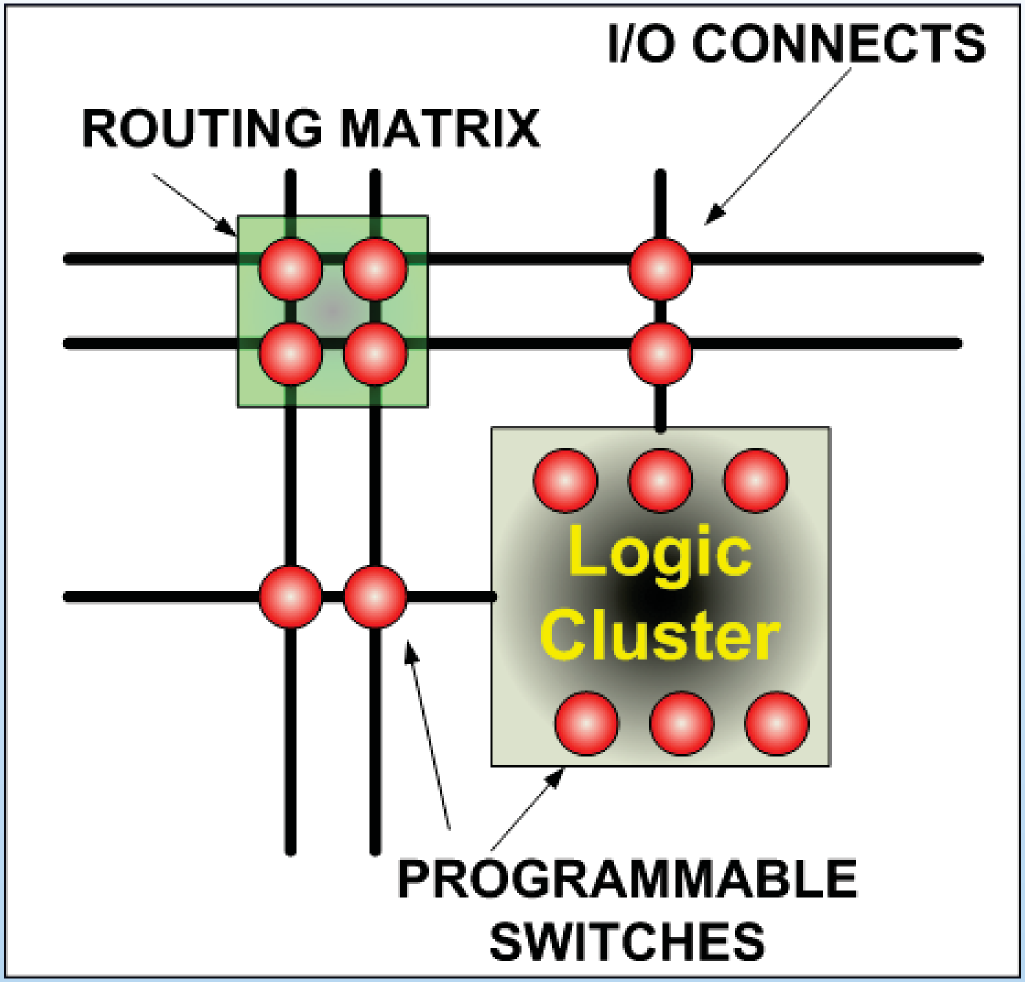
Hardware design language (HDL)



FPGA Configuration (Storage of User Design Mapping)

HDL → **FPGA MAPPING** → **Configuration**

- ***Configuration Defines:*** *Arrangement of pre-existing logic via programmable switches.*
  - *Functionality (logic cluster) and*
  - *Connectivity (routes)*
- ***Programmable Switch Types:***
  - ***Antifuse:*** *One time Programmable (OTP),*
  - ***SRAM:*** *Reprogrammable (RP), or*
  - ***Flash:*** *Reprogrammable (RP).*

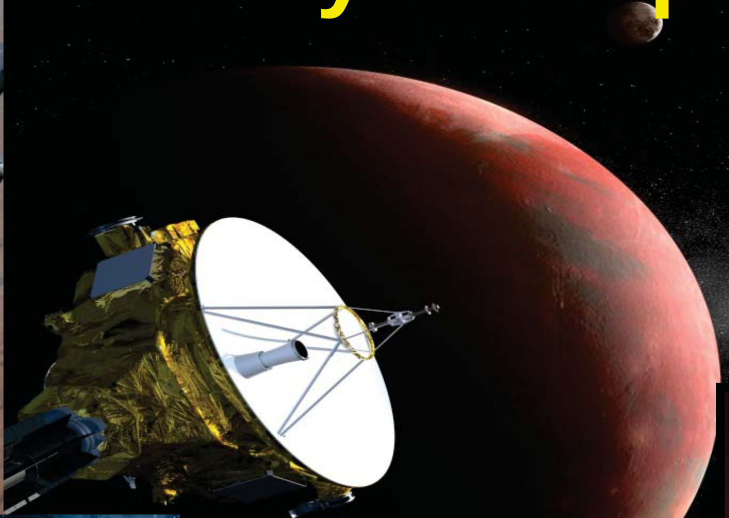


FPGA’s And Critical Applications

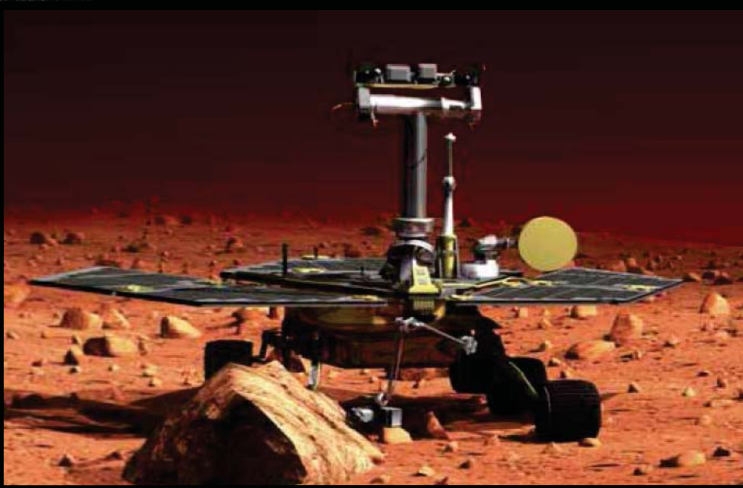
Common FPGA Applications

- **Controllers,**
- **Dataflow and interface adaptation,**
- **Digital signal processing (DSP),**
- **Software-defined radio,**
- **ASIC prototyping,**
- **Medical imaging,**
- **Robotic control (vision, movement, speech, etc.,…)**
- **Cryptology,**
- **Nuclear plant control,**
- **The list goes on…**

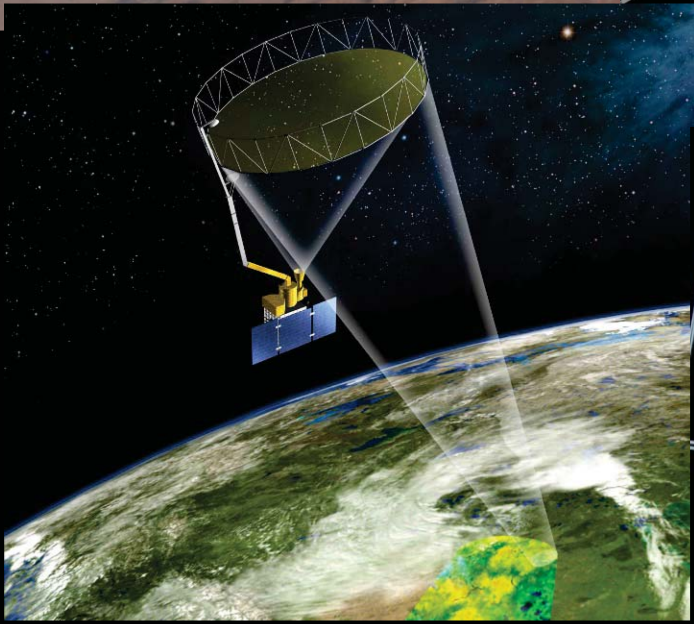
Common Applications Example 1: Military and Space



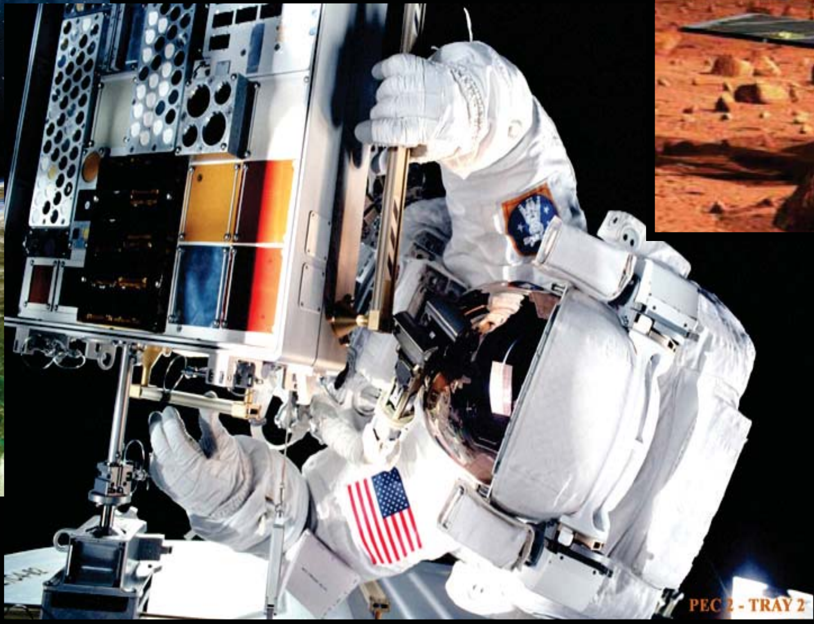
New Horizons
Pluto and Beyond



Mars Rover

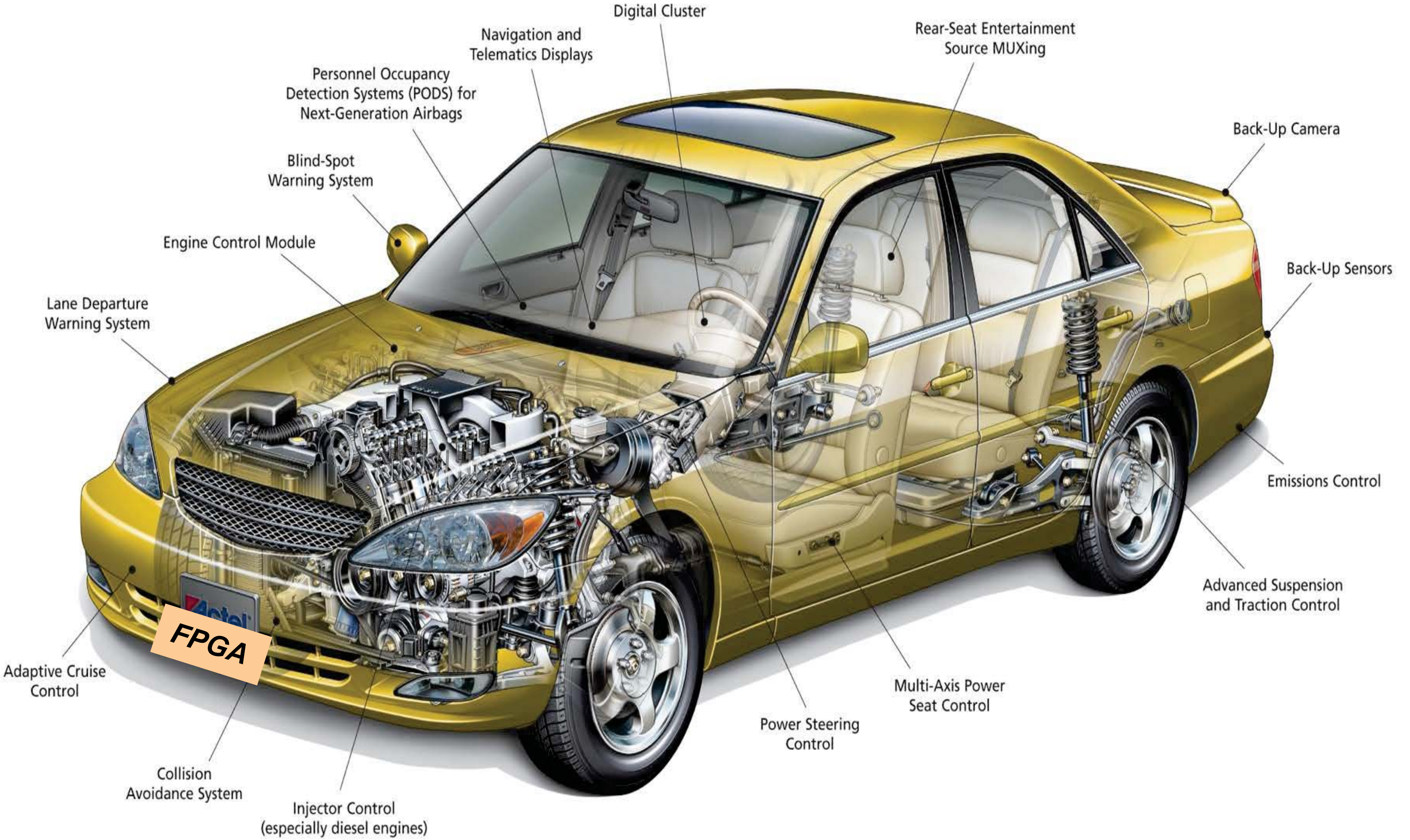


Soil Moisture
Active Passive



Spacecube:
International
Space Station

Common Applications Example 2: Terrestrial



http://www.eetimes.com/document.asp?doc_id=1305894

Concerns for using FPGA Devices in Critical Applications

- **Safety**: can circuits or humans be damaged or hurt?
- **Reliability** : will the device operate as expected?
- **Availability**: how often will the system operate as expected?
- **Recoverability**: if the device malfunctions, can the system come back to a working state?
- **Trust**: Will the insertion of the device compromise security?

Critical applications will want to avoid disaster.

Sources of FPGA Failures

Packaging and mounting

Negative bias temperature instability (NBTI)

Hot carrier injection (HCI),

Poor design choices

Electromigration (EM)

Single event effects (SEEs)

Dielectric breakdown (DB)

Environmental stress

Total ionizing dose (TID)

Transistor switching stress

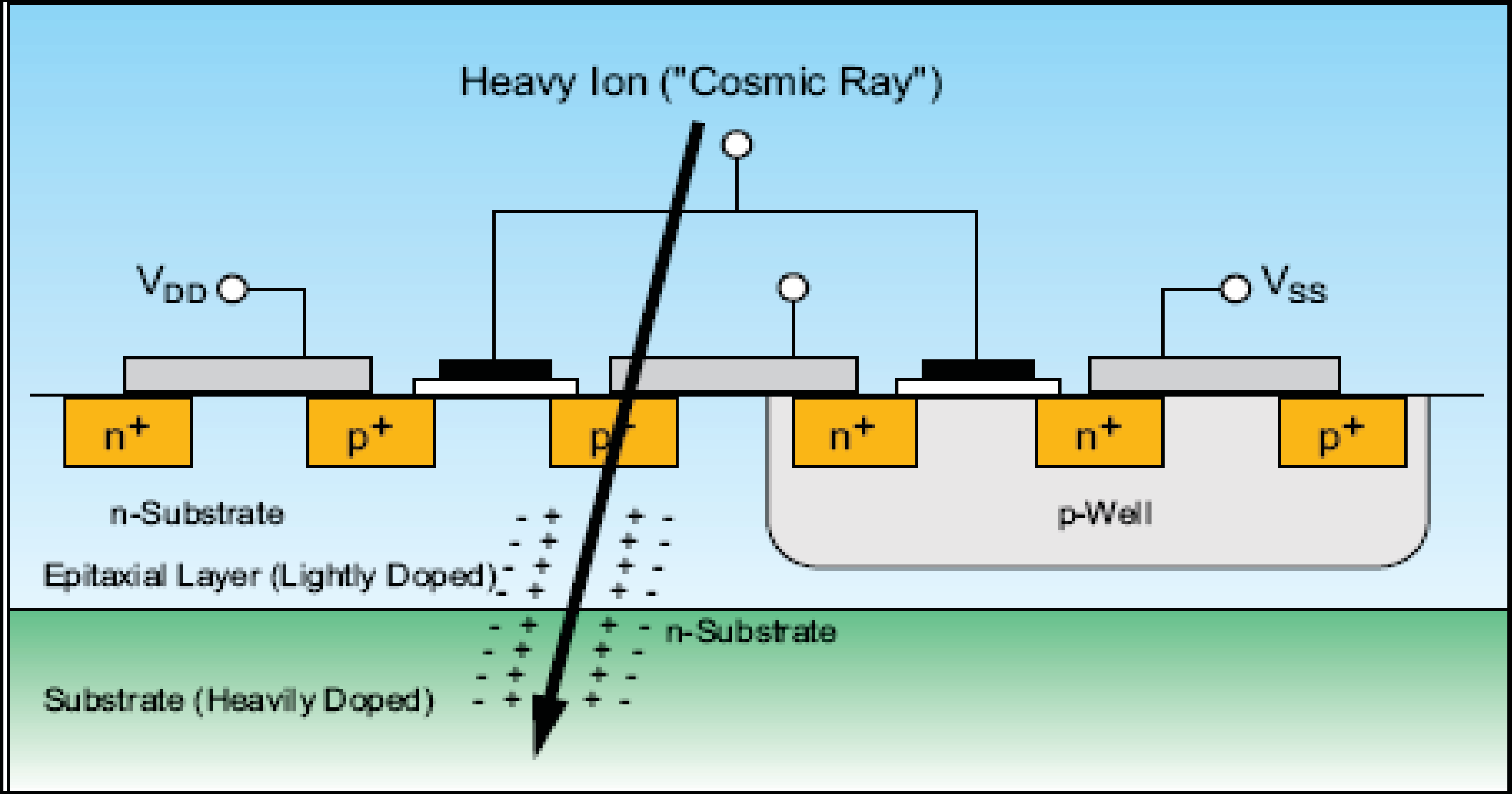
Lack of verification

How To Protect A System from Failure

- **Investigate failure modes – understand risk:**
  - **Reliability testing (temperature, voltage, mechanical, and logic switching stresses).**
  - **Radiation testing: Single event effects (SEE) and total ionizing dose (TID).**
- **Add redundancy:**
  - **Replication with correction.**
  - **Replication with detection. Requires recovery:**
    - **Switch to another device,**
    - **Try to recover state,**
    - **Start over,**
    - **Alert,**
    - **Do nothing… die.**
- **Add filtration: e.g., Finite impulse response (FIR) filters or Constant false alarm rate filter (CFAR).**
- **Add masking: Protect system operation from failures.**

Single Event Upsets (SEUs) and FPGA Devices

- **Although there are many sources of FPGA malfunction, this presentation will focus on SEUs as a source of failure.**



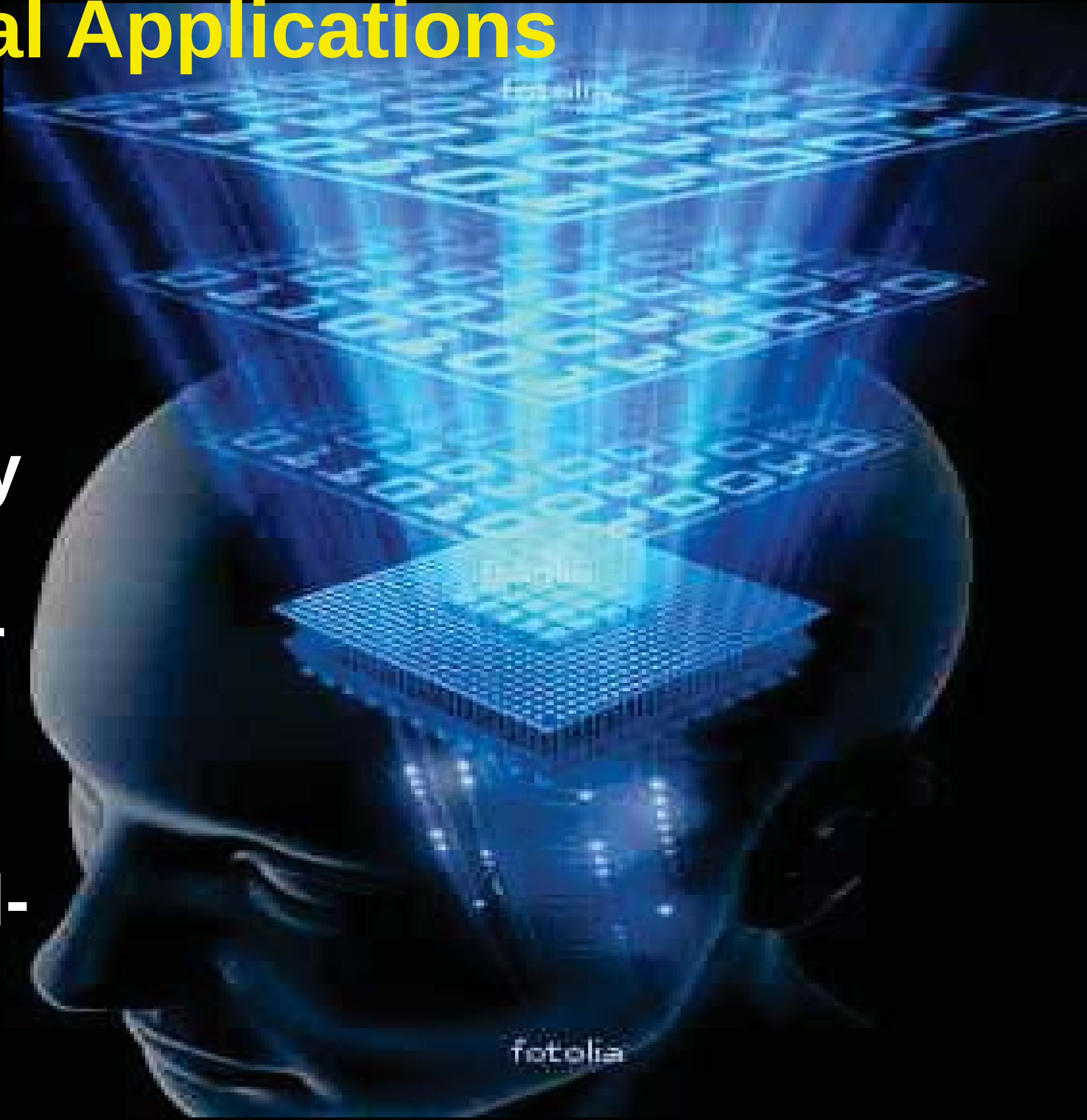
Implications of SEUs to FPGA Applications

- **Ionizing particles cause upsets (SEUs) in FPGAs.**
- **Each FPGA type has different SEU error signatures:**
  - **Temporary glitch (transient),**
  - **Change of state (incorrect state machine transitions),**
  - **Global upsets: Loss of clock or unexpected reset,**
  - **Configuration corruption. This includes route breakage (no signal can get through) – can be overwhelming.**
- **The question is how to avoid system failure and the answer depends on the following:**
  - **The system's requirements and the definition of failure,**
  - **The target FPGA and its surrounding circuitry susceptibility,**
  - **Implemented fail-safe strategies,**
  - **Reliable design practices,**
  - **Radiation environment.**

SEE Go-no-Go: Single Event Hard Faults and Common Terminology

- **Single Event Latch Up (SEL): Device latches in high current state:**
  - **Has been observed in FPGA devices that are currently on the market.**
  - **Some missions choose to use the devices and design around the SEL.**
- **Single Event Burnout (SEB): Device draws high current and burns out.**
  - **Not observed in FPGA devices that are currently on the market.**
- **Single Event Gate Rupture: (SEGR): Gate destroyed typically in power MOSFETs.**
  - **Not observed in FPGA devices that are currently on the market.**

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.

Mitigation

- Error Masking vs. Error Correction… there's a difference.
- Mitigation can be:
  - **User inserted:** part of the actual design process.
    - User must verify mitigation… Complexity is a RISK!!!!!!!!
  - **Embedded:** built into the device library cells.
    - User does not verify the mitigation – manufacturer does.
- Mitigation should reduce error…
  - Generally through redundancy.
  - Incorrect implementation can increase error.
  - Overly complex mitigation cannot be verified and incurs too high of a risk to implement.

Differentiating Fail-Safe Strategies:

- Detection:
  - Watchdog (state or logic monitoring).
  - Simplistic Checking … Complex Decoding.
  - Action (correction or recovery).
- Masking (does not mean correction):
  - Not letting an error propagate to other logic.
  - 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.

Availability versus Correct Operation

- **Requirements must be satisfied.**
- **What is your expected up-time versus down-time (availability)?**
- **Is correct operation well defined? Unambiguous!**
- **Is system failure well defined? Unambiguous!**
- **Can availability and correct operation be deterministic regardless of error signature?**
- **Availability:**
  - **Flushable designs: systems than can be reset or are self-correcting. Availability is affected during reset or correction time (down-time). However, downtime is tolerable as defined by system requirements.**
  - **Non-flushable designs: System requirements are strict and require minimal downtime. Usage of resets are required to be kept at a minimum.**

Detection and Recovery

- **Not all mitigation schemes require detection.**
- **Questions/Considerations:**
  - **If your scheme requires detection:**
    - **Can the system detect all error signatures?**
    - **Can the system detect all error signatures fast enough?**
    - **Do different errors require different recovery schemes… can the system accommodate.**
  - **How are you going to verify the detection and recovery?**
  - **How much downtime will there be during recovery**

Availability = detection + recovery time – masked error time

"Yes or "We know it will work" are not good enough answers: Ask how and if the scheme has been verified!

To be presented by Melanie Berg via WebEx at SERRESSA 2015 International School on the Effects of Radiation on Embedded Systems for Space Applications, Puebla, Mexico, November 30 to December 4, 2015.

SEUs and FPGA Variations

- FPGA susceptibilities (error signatures) vary per FPGA type.
- How does a project manage and protect against FPGA susceptibilities? (mitigation schemes will change based on FPGA type).
- The most efficient solution will be based on understanding:
  - SEE theory,
  - FPGA SEE susceptibility (per FPGA type),
  - Proven mitigation strategies per FPGA type,
  - Validation and verification of implemented mitigation strategies, and
  - Limitations of tools and/or mitigation schemes.

Redundancy Is Not Enough

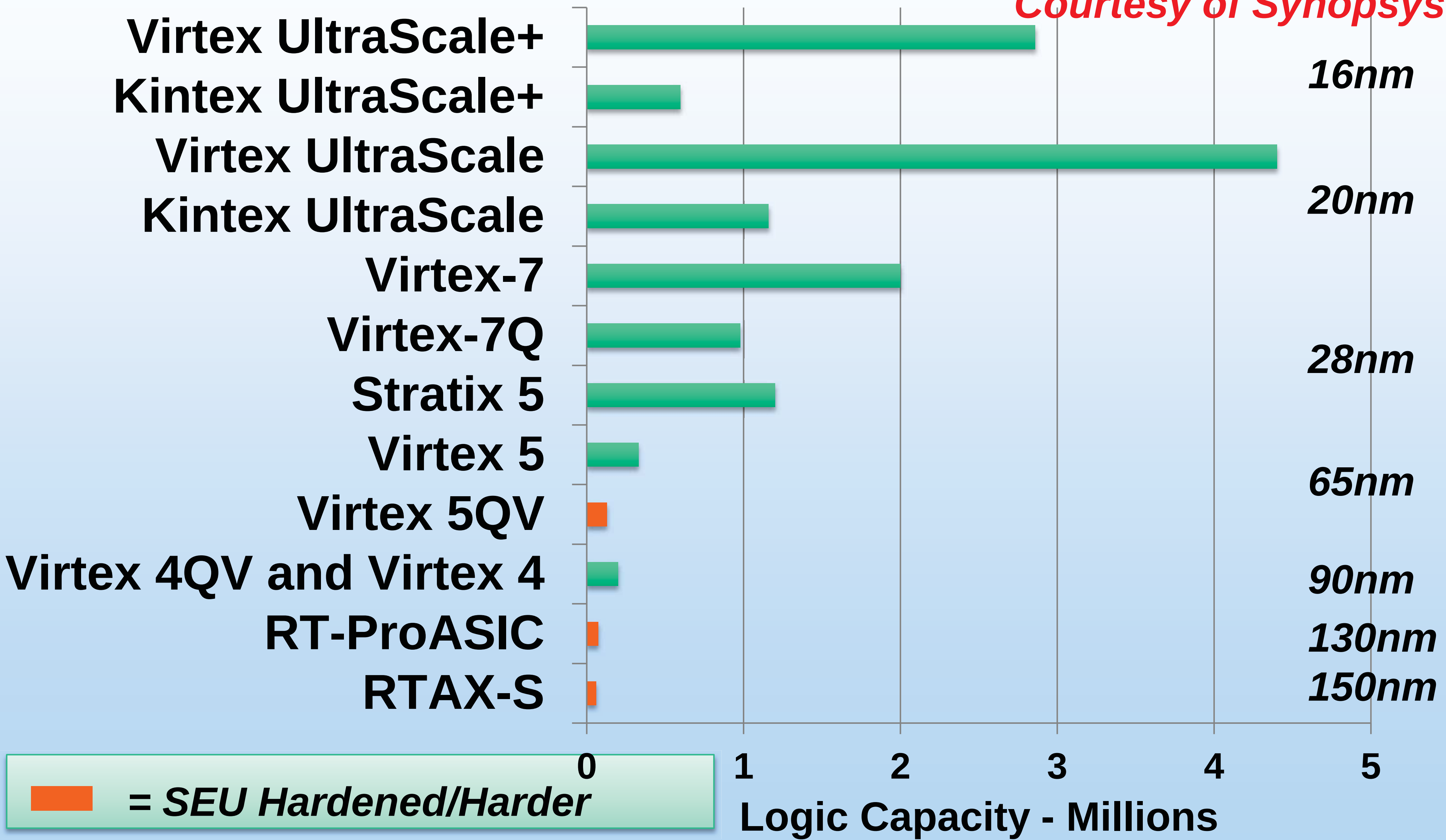
- **Just adding redundancy to a system is not enough to assume that the system is well protected.**
- **Concerns that must be addressed for a critical system expecting redundancy to cure all (or most):**
  - **How is the redundancy implemented?**
  - **What portions of your system are protected? Does 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?**

Radiation Hardened (per SEU) versus Commercial FPGA Devices

- **A radiation hardened (per SEU) FPGA is a device that has embedded mitigation.**
- **Radiation hardened FPGA devices are available to users. They make the design cycle much easier!**
- **They are considered hardened if:**
  - **Configuration susceptibility is reduced to an acceptable rate.**
  - **Generally, less than one node per 1x10^{-8} days.**
  - **Be careful: with millions of nodes, this can translate into 1 or two configuration failures per year.**
  - **However, if the node isn't being used, then your circuit may not be affected by the failure.**

Radiation Hardened versus Commercial FPGA Device Geometries And Gate Count

As Geometries Get Smaller, More Gates Are Available for Mitigation



Courtesy of Synopsys

FPGA Devices Listed by Configuration Type (Not All Are Included in The List): Embedded Mitigation

Manufacturer	Configuration Type	Short List of Device Families	Embedded Mitigation
Altera	SRAM	Stratix	No
Microsemi	Antifuse	RTAX, RTSXS	Clocks +DFFs (configuration is already hardened by nature)
Microsemi	Flash	ProASIC3	Configuration is already hardened by nature.
Xilinx	SRAM	Virtex, Kintex	No
Xilinx	Hardened SRAM	Virtex V5QV	Configuration + DICE DFFs + SET filters

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.

FPGA Devices Listed by Configuration Type (Not All Are Included in The List): Susceptibility

Configuration Type	Short List of Device Families	Embedded Mitigation	Most Susceptible Components
SRAM	Stratix, Virtex, Kintex	No	Configuration
Antifuse	RTAX, RTSXS	DFFs and clocks (configuration is already hardened by nature)	Combinatorial logic (however susceptibility considered low)
Flash	ProASIC3	Configuration is already hardened by nature.	DFFs and clocks
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.

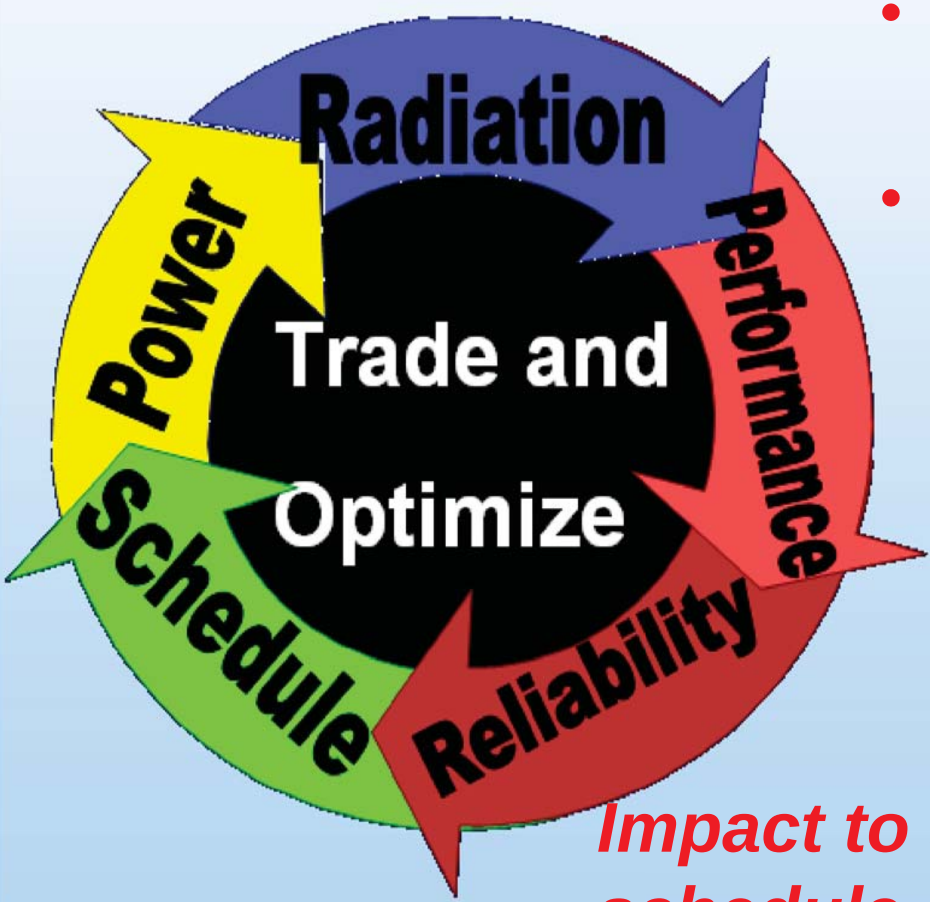
NASA and other Government Agency FPGA Device Selection for Critical Applications

- Currently, the most common FPGA devices used for NASA driven critical space applications are anti-fuse.
- This is also true for other government agencies.
- However, due to cost of implementation and robustness of design, SRAM-based FPGAs are becoming more popular.
- The usage of SRAM-based FPGA devices introduces a variety of challenges for critical operations because their SEU susceptibility and reduced security.

Preliminary Design Considerations for Mitigation And Trade Space

Determine Most Susceptible Components:

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



- **Does the designer need to add mitigation?**
- **Will there be compromises?**
  - **Performance and speed,**
  - **Power,**
  - **Schedule**
  - **Mitigating the susceptible components?**
  - **Reliability (working and mitigating as expected)?**

Impact to speed, power, area, reliability, and schedule are important questions to ask.

Fail-safe Strategies for Single Event Upsets (SEUs)

- The following slides will demonstrate commonly used mitigation strategies for FPGA devices.
- What you should learn:
  - The differences between FPGA 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

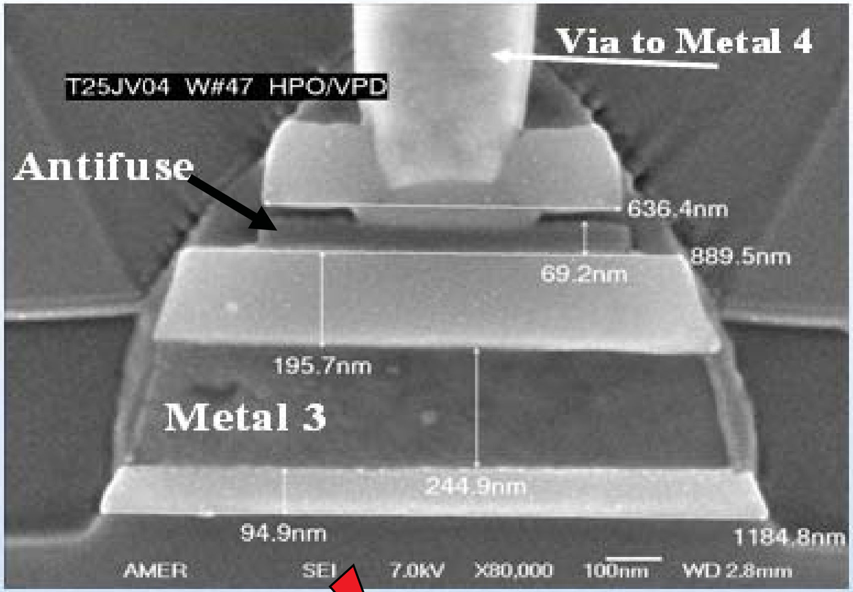
Single Event Upsets and FPGA Configuration

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

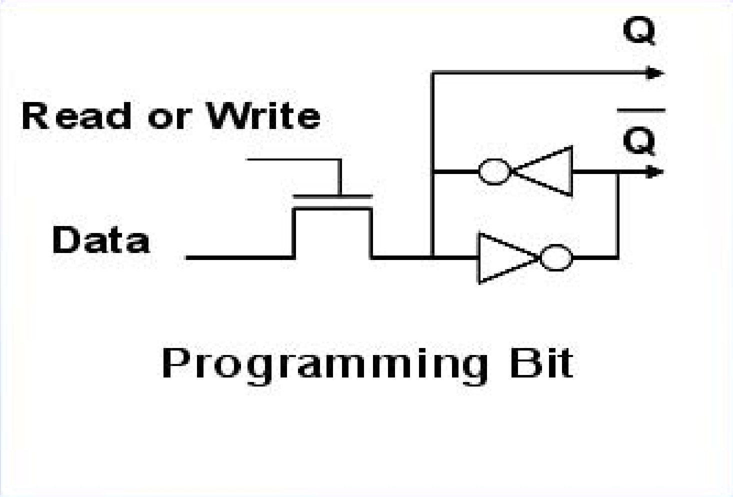
To be presented by Melanie Berg via WebEx at SERRESSA 2015 International School on the Effects of Radiation on Embedded Systems for Space Applications, Puebla, Mexico, November 30 to December 4, 2015.

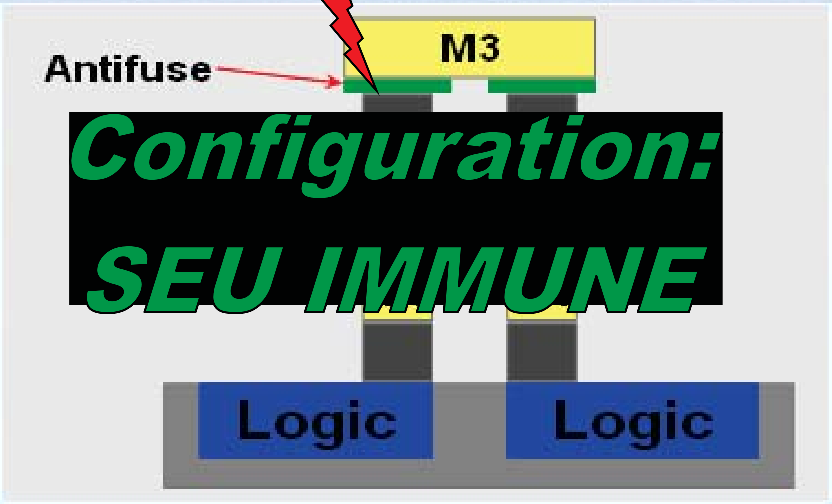
Programmable Switch Implementation and SEU Susceptibility

ANTIFUSE *(one time programmable)*

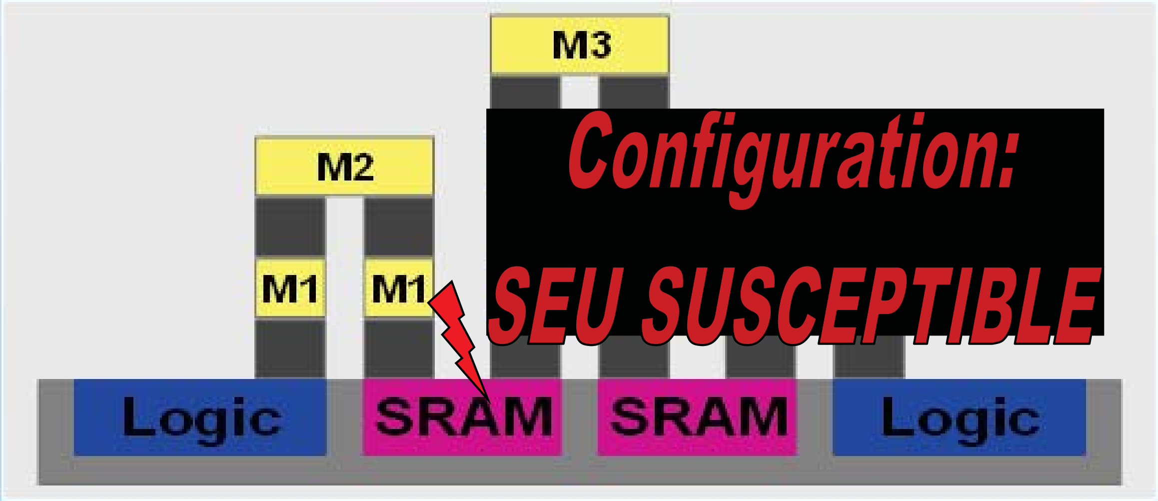


SRAM *(reprogrammable)*





Configuration: SEU IMMUNE



Configuration: SEU SUSCEPTIBLE

Configuration SEU Test Results and the REAG FPGA SEU Model

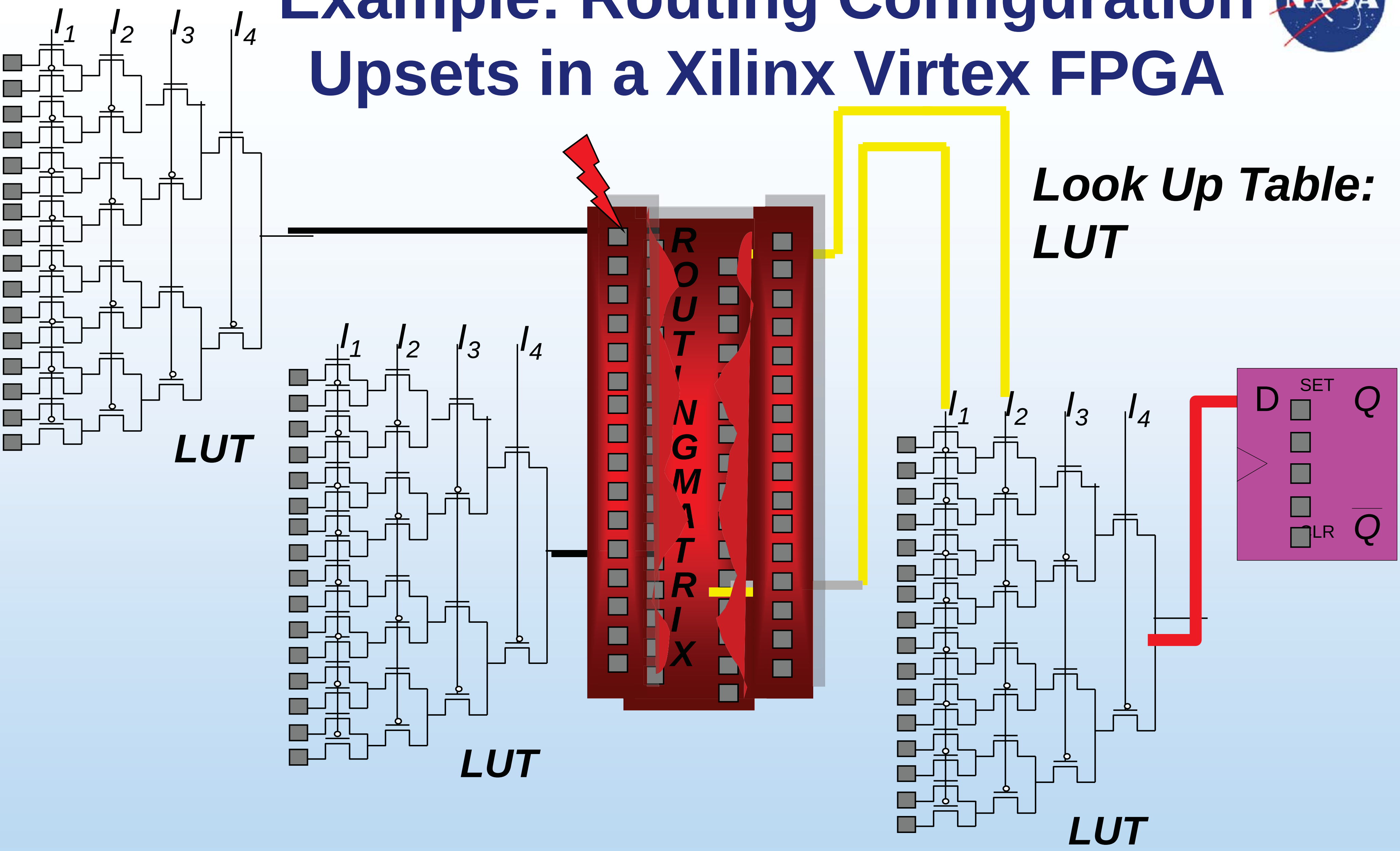
$$P(fs)_{error} \propto P_{Configurat\ ion} + P(fs)_{functional\ Logic} + P_{SEFI}$$

FPGA Configuration Type	REAG Model
Antifuse	$P(fs)_{error} \propto P(fs)_{functionalLogic} + P_{SEFI}$
SRAM **(non-mitigated)**	$P(fs)_{error} \propto P_{Configuration}$
Flash	$P(fs)_{error} \propto P(fs)_{functionalLogic} + P_{SEFI}$
Hardened SRAM	$P(fs)_{error} \propto P_{Configuration} + P(fs)_{functionalLogic} + P_{SEFI}$

What Does The Last Slide Mean?

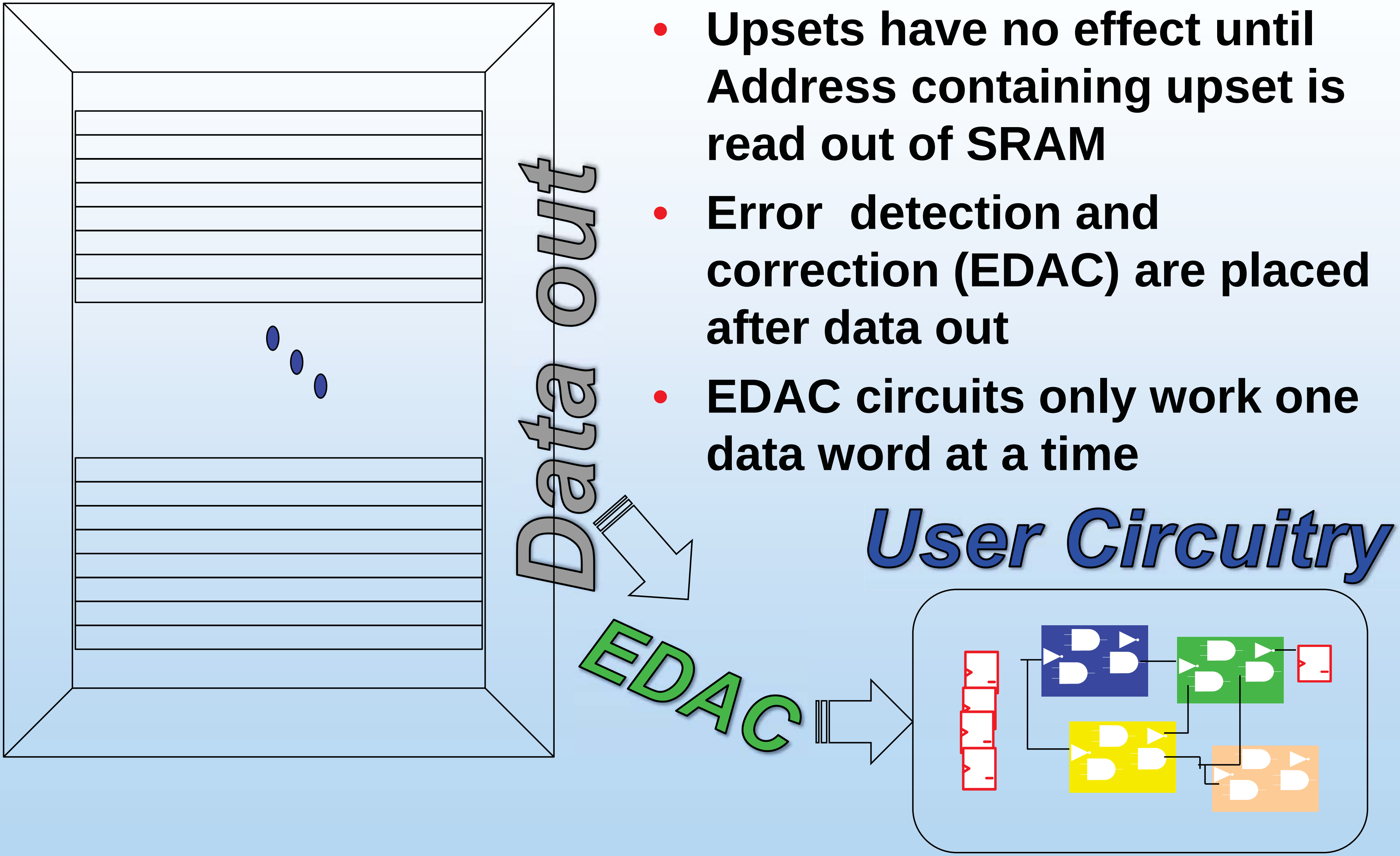
FPGA Configuration Type	**Susceptibility** **Data-path: Combinatorial Logic (CL) and Flip-flops (DFFs);** **Global: Clocks and Resets;** **Configuration**
Antifuse	Configuration has been designated as hard regarding SEEs. Susceptibilities only exist in the data paths and global routes. However, global 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).

Example: Routing Configuration Upsets in a Xilinx Virtex FPGA



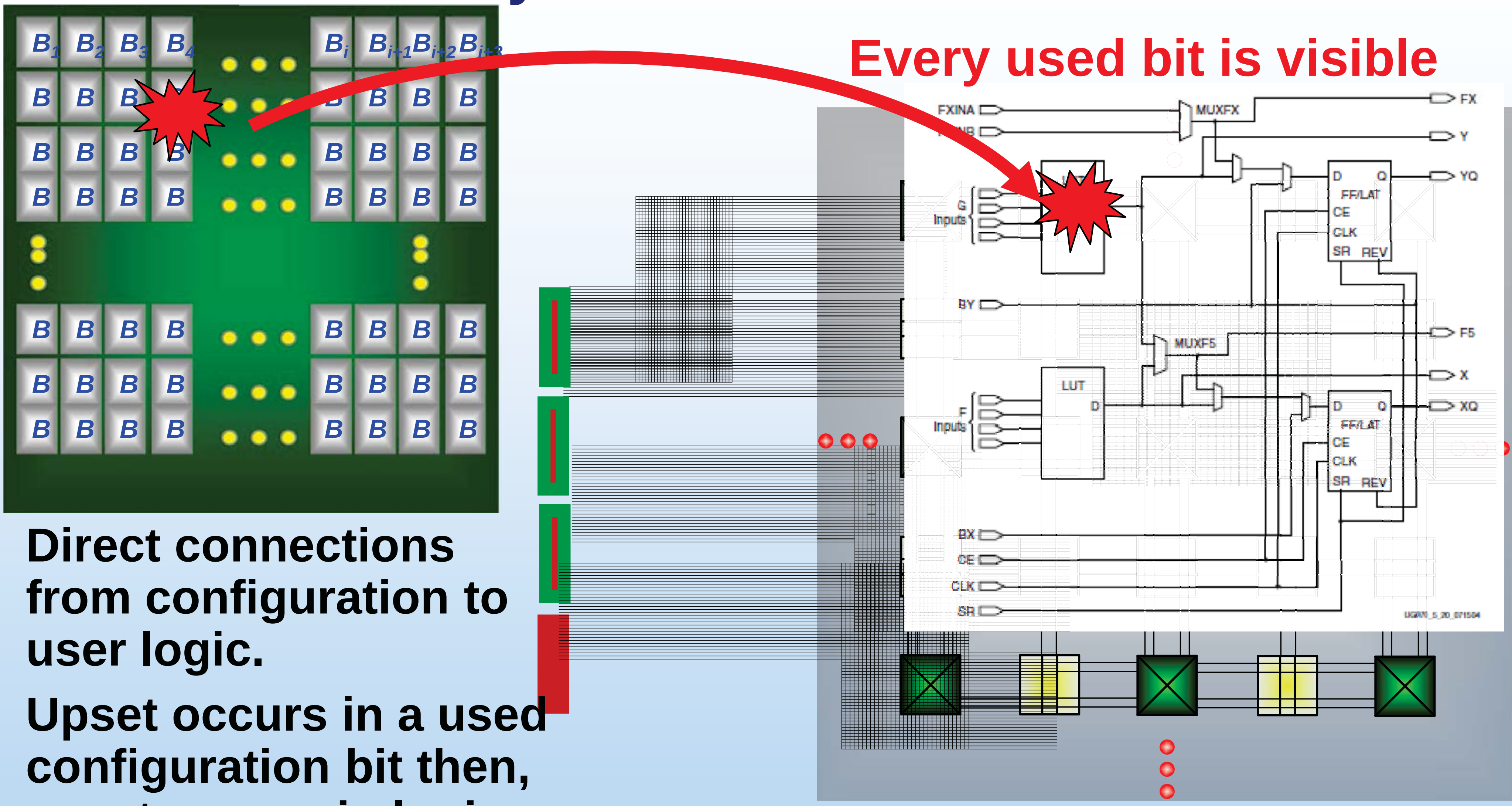
Because multiple paths can pass through the routing matrix, this configuration can be catestrophic – i.e., break simple mitigation

Traditional SRAM … One Data Word at a Time



- Upsets have no effect until Address containing upset is read out of SRAM
- Error detection and correction (EDAC) are placed after data out
- EDAC circuits only work one data word at a time

Configuration SRAM is NOT Utilized the Same Way as Traditional SRAM



Every used bit is visible

- Direct connections from configuration to user logic.
- Upset occurs in a used configuration bit then, upset occurs in logic.
- We're not dealing with data words anymore. Traditional SRAM EDAC schemes don't quite apply for configuration SRAM

Scrubbers: Blind versus Read-back

Blind Scrubber

- **Write golden configuration into configuration**
- **Scrub cycle in the order of *ms***
- **Pros: simple, less area and power, no need for additional non-volatile memory, very fast (great for accelerated testing)**
- **Cons: Write pointer can get hit during writing and write bad data into configuration- however, insignificant probability of occurrence (proven in heavy ion SEU testing)**

Read-back

- **Read configuration, calculate correct data; if there is an upset, write corrected data.**
- **Scrub cycle in the order of *s***
- **Pros, only writes if there is an upset**
- **Cons, additional non-volatile memory required; slow (only a problem for accelerated testing); takes more area and power; Correction scheme can break (e.g. be limited to detecting and correcting one upset); Consequently, upon an MBU can write bad data to configuration – this has been proven via heavy-ion testing.**

Scrubbers: Internal versus External (1)

- Internal and external scrubbers are used to fix configuration bits:
  - 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 an separate device .
- External scrubbers are usually implemented in anti-fuse FPGA devices.
- Internal scrubbers are obviously more susceptible than external scrubbers.

Scrubbers: Internal versus External (2)

- **Internal scrubbers are usually implemented as read-back. Remember read-back scrubbers can break and write bad frames into the configuration due to MBUs.**
- **Although configuration memory interleaving has been used in the newer SRAM-based FPGAs, a significant number of Multiple Bit Upsets (MBUs) have been observed via Naval Research Laboratory (NRL) laser testing.**
  - **Could be because of laser spot size.**
  - **More testing is expected to be performed early 2016.**

Differentiate Scrubbing for Space Applications and Scrubbing for Radiation Testing

Space Application

- Only scrub if there is mitigation
- Make scrubber simple to reduce project risk
- Do not scrub constantly – not necessary and not good for the system
- Single error correction double error detection (SECDED) scrubbers may not work well due to multiple bit upsets (MBUs)
- Blind scrubbing is the simplest scheme yet read-back will also work

Accelerated SEU Testing

- We must scrub!
- Particles cannot overtake the scrubber – i.e., scrubber must be fast enough to stop fast accumulation of configuration SEUs – SCRUB CONSTANTLY
- SECDED scrubbing schemes do not work well during accelerated testing because of MBUs and accumulation
- Generally no time for read-back of configuration – hence blind scrubber is the best fit for accelerated testing

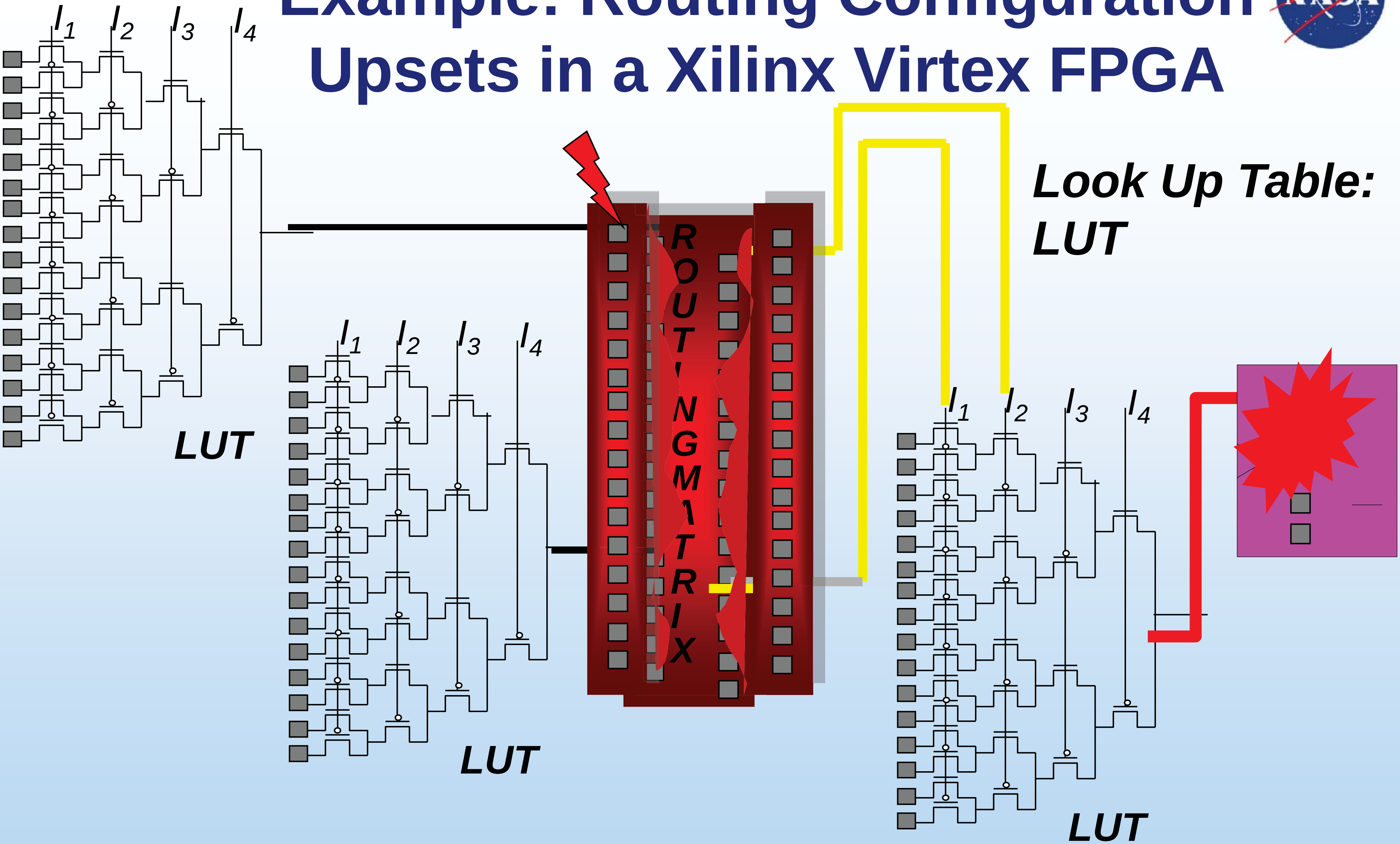
Warning!

- **Fixing a configuration bit does not mean that you have fixed the state in the functional logic path.**
- **In order to guarantee that the functional logic is in the expected state after the configuration bit is fixed, either the state must be restored or a reset must be issued.**

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.

Single Event Upsets in an FPGA's Functional Data Path and Fail-Safe Strategies

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

To be presented by Melanie Berg via WebEx at SERRESSA 2015 International School on the Effects of Radiation on Embedded Systems for Space Applications, Puebla, Mexico, November 30 to December 4, 2015.

Data-path SEUs and Their Affect At The System Level

- A system implemented in an FPGA is a cascade of sequential and combinatorial logic.
- The occurrence of an SET or SEU does not definitively cause system error.
- Probability of a system error due to an SEU depends on many factors:
  - Probability of fault generation in a gate (SET or SEU).
  - Probability of error propagation – will the SET or SEU force the system's next state to be incorrect?

Probability of Error Propagation in A Data-Path

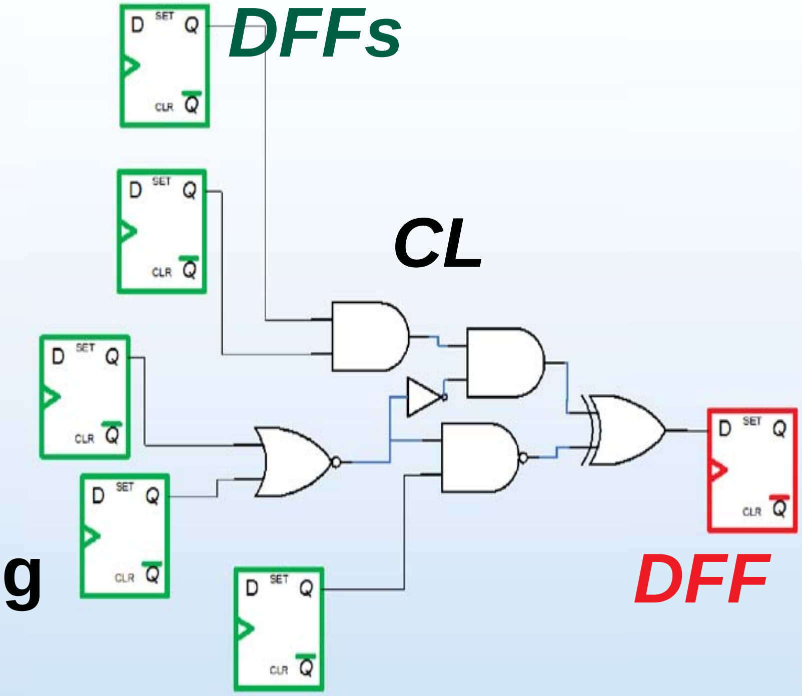
Upsets usually occur between clock cycles: Can cause a system-level malfunction if the SET or SEU will force the system's next state to be incorrect.

- **Capacitive filtration: data-path capacitance can stop transient upset propagation; e.g.:**
  - **Routing metal or heavy loading.**
  - **If a transient doesn't reach a sequential element, then it most likely will not cause a system upset.**
- **Logic masking:**
  - **Redundancy and mitigation of paths can stop upset propagation.**
  - **Turned off paths from gated logic can stop upset propagation.**
- **Temporal delay: path delays can block temporary SEUs from disturbing next state calculation.**

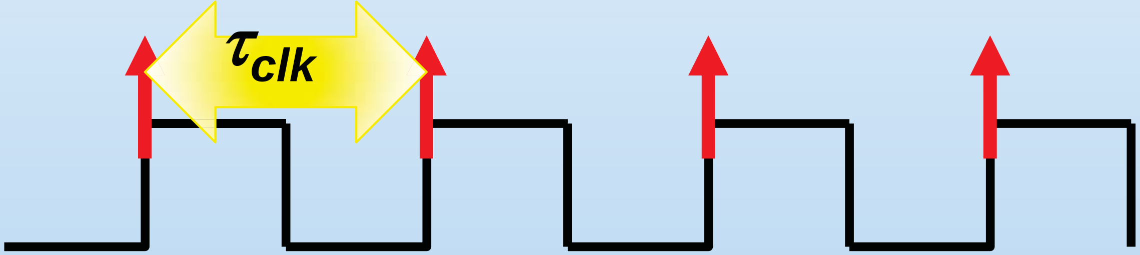
Data-path SEU Susceptibility and Analysis : the NASA Electronics Parts and Packaging (NEPP) FPGA Model

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, July 2012.

Background: Synchronous Design Data Path – Sample and Hold



- **Synchronous design components:**
  - **Edge Triggered Flip-Flops (DFFs),**
  - **Clocks and resets (global routes),**
  - **Combinatorial Logic (CL).**
- **All DFFs are connected to a clock.**
- **DFFs sample their input at the rising edge of clock.**



Clock Period

$$\tau_{clk} = \frac{1}{fs}$$

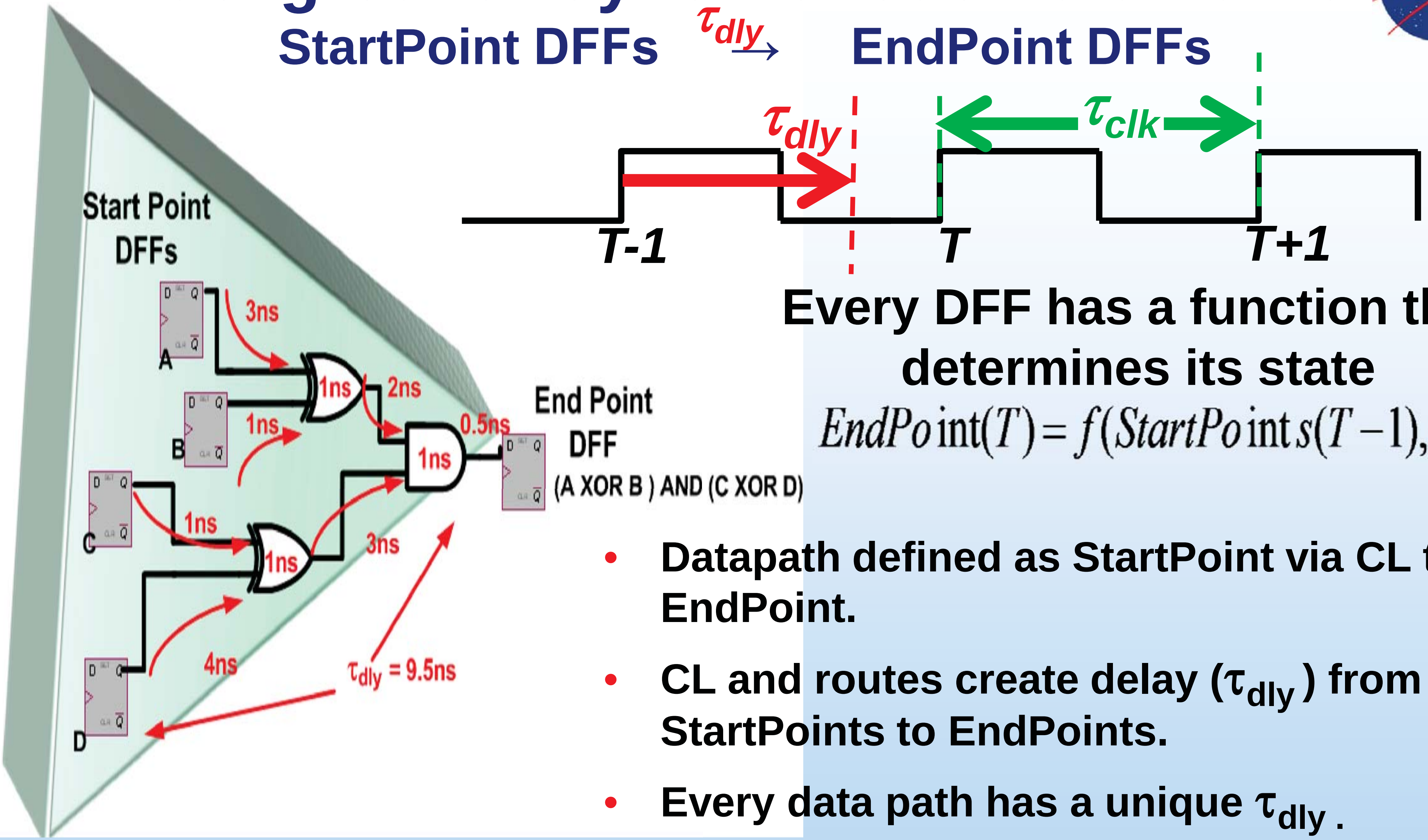
Frequency

- **CL compute between clock edges.**

Designs are complex – We modularize for simplicity

Background: Synchronous Data Paths:

StartPoint DFFs $\xrightarrow{\tau_{dly}}$ EndPoint DFFs



Every DFF has a function that determines its state

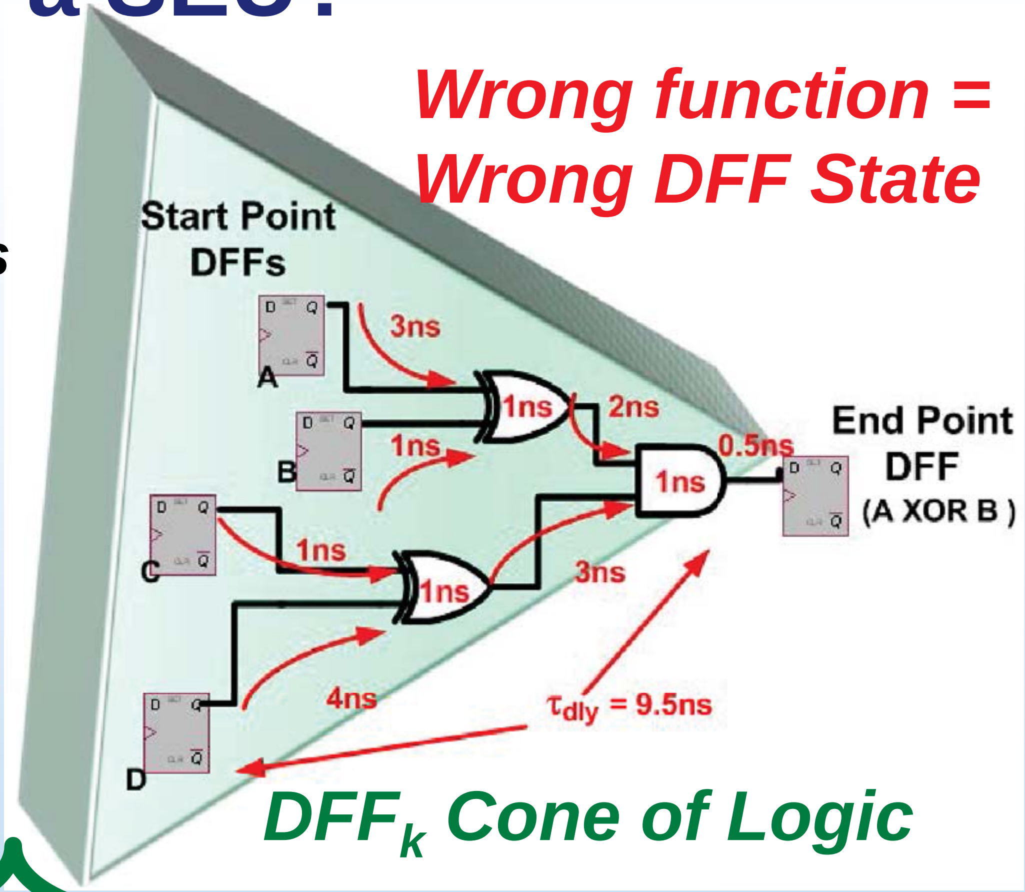
$$EndPoint(T) = f(StartPoints(T-1), CL)$$

- **Datapath defined as StartPoint via CL to EndPoint.**
- **CL and routes create delay (τ_{dly}) from StartPoints to EndPoints.**
- **Every data path has a unique τ_{dly}.**
- **τ_{dly} is calculated using Static Timing Analysis (STA) design tools.**

Modularization: Every DFF has a unique cone of logic

How can a DFF Contain an Incorrect State from a SEU?

- ***DFFs have various modes of reaching a bad state due to SEUs.***
- ***Attribute some modes to EndPoints and some to StartPoints.***



Wrong function = Wrong DFF State

DFF_k Cone of Logic

We make a clear distinction between DFF SEUs based on Clock state and Capture.

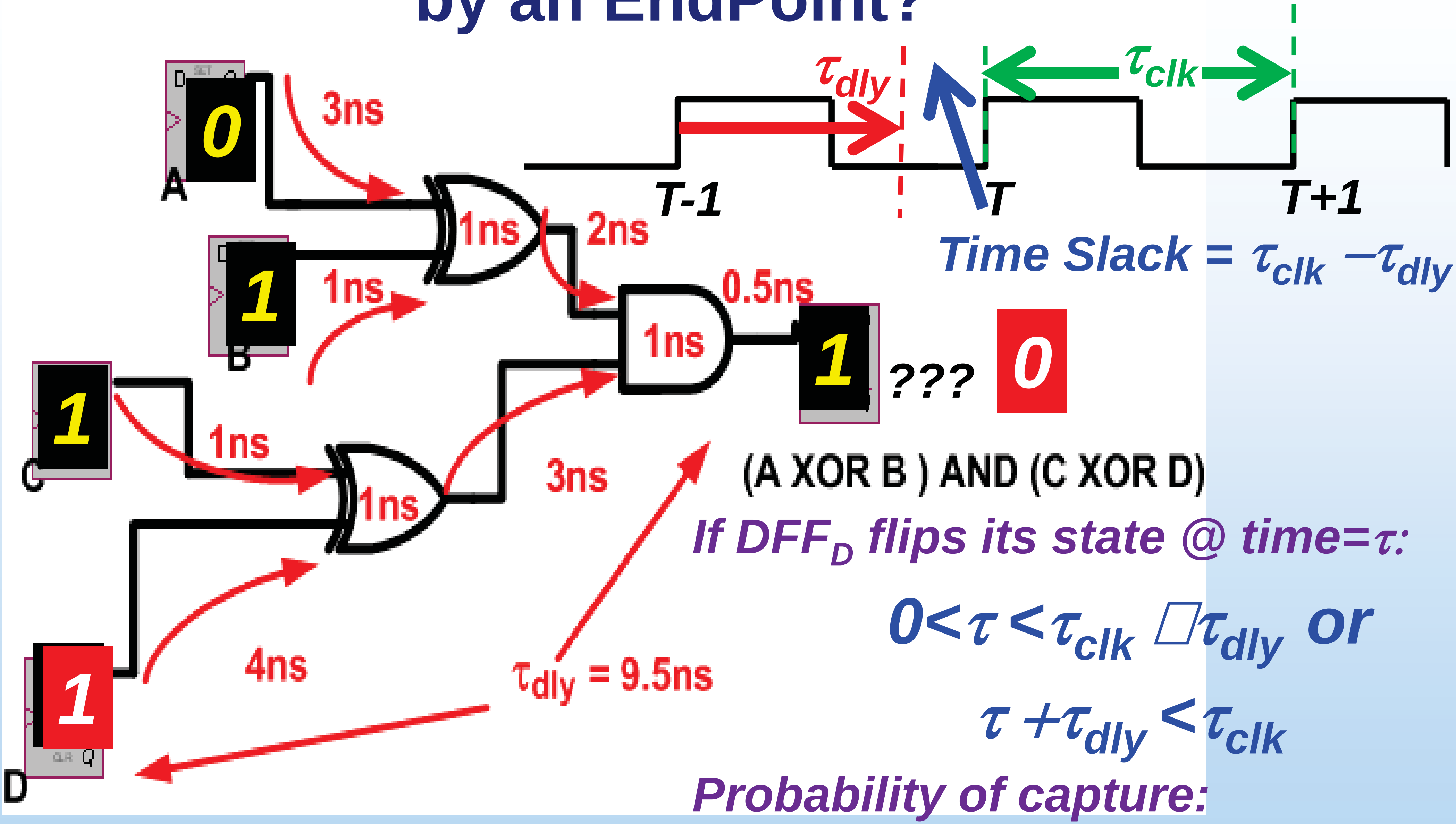
$\forall_{EndPoint\ DFF}$ ***EndPoint DFF SEUs + StartPoint DFF SEUs + CL SETs***

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.

How Does a StartPoint SEU get Captured by an EndPoint?



Time Slack = $\tau_{clk} - \tau_{dly}$

(A XOR B) AND (C XOR D)

If DFF$_D$ flips its state @ time=τ:

$$0<\tau<\tau_{clk} \square \tau_{dly} \text{ or}$$

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

Probability of capture:

$$1-(\tau_{dly}/\tau_{clk}) = 1-\tau_{dly}fs$$

Details of Capturing StartPoint DFFs

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

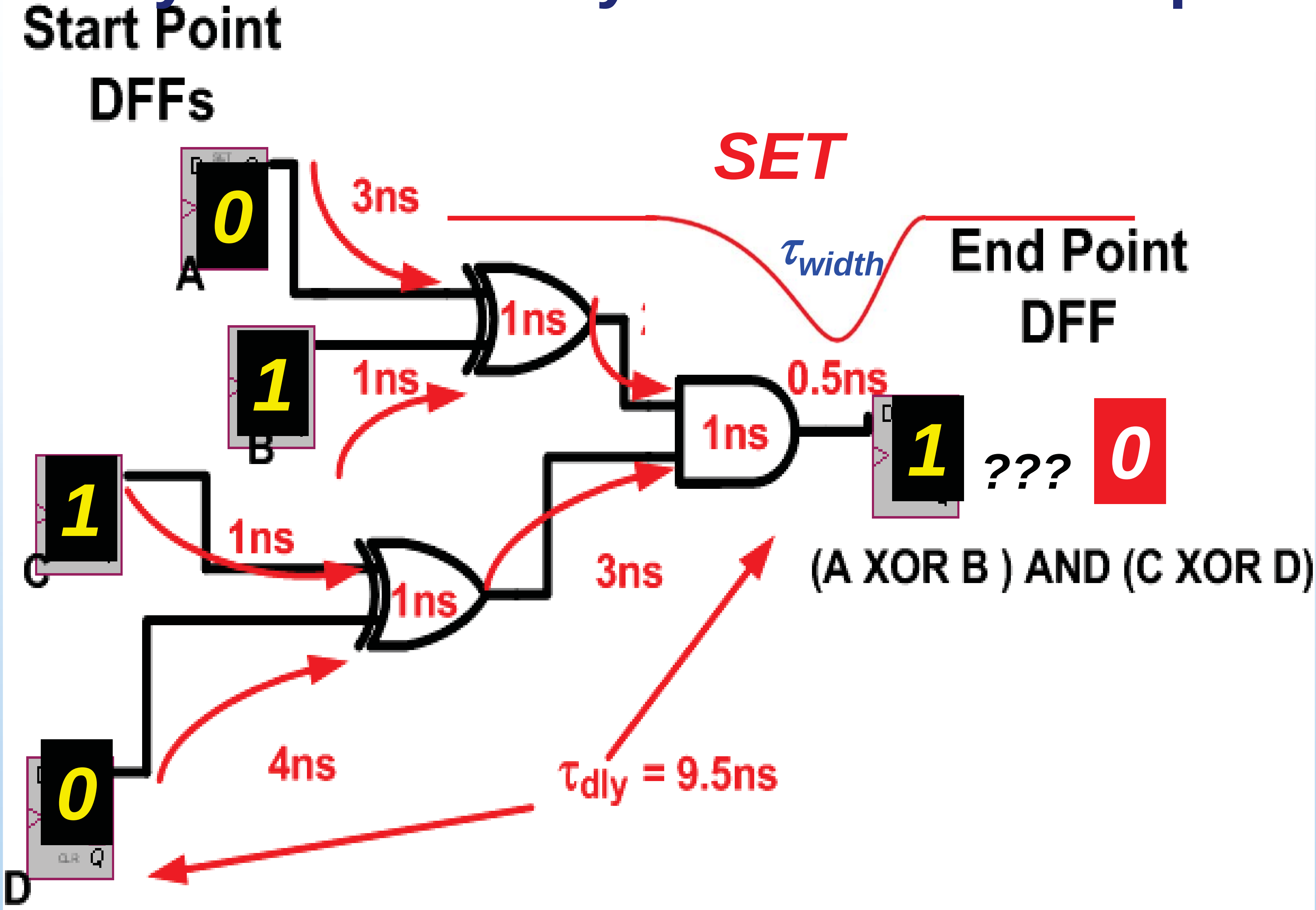
Upset generated internally to DFF between clock edges

Design Topology and Temporal Masking

Design Topology and Logic Masking

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

Synchronous System: CL SET Capture



To be presented by Melanie Berg via WebEx at SERRESSA 2015 International School on the Effects of Radiation on Embedded Systems for Space Applications, Puebla, Mexico, November 30 to December 4, 2015.

Details of CL SET Capture

$$\forall_{DFF}\left(\sum_{i=1}^{\#Combinator\,ialCells}\left(P_{gen(i)}P_{prop(i)}P_{\log ic}\tau_{width(i)}fs\right)\right)$$

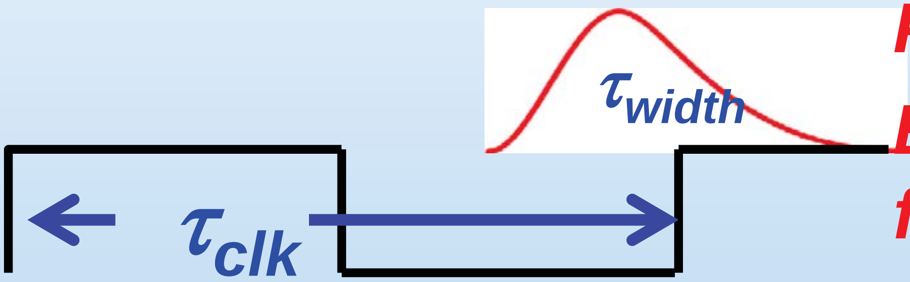
Generation

Logic Masking

Propagation: Electrical Masking from routes and gate cut-off frequencies

Width of SET relative to clock period τ_{clk}

SET



- SET Generation (P_{gen}) occurs between clock edges
- EndPoint DFF captures the SET at a clock edge
  - Increase frequency – increase probability of capture
  - Increase CL – increase probability of capture

NEPP FPGA Model: Putting it All Together – Analyzed Per Particle Linear Energy Transfer (LET)

$$\sum_{k=1}^{\#EndPoint\ DFFs} \overset{\text{EndPoint Logic Masking}}{P_{logic(k)}} * \left(\alpha P(fs)_{DFFSEU(k)} + \text{EndPoint} \quad \sum_{j=1}^{\#StartPoint\ DFFs} (\beta P(fs)_{DFFSEU(J)}(1-\tau_{dly(j)} fs)) * P_{logic(j)}) + \text{StartPoints} \quad \sum_{i=1}^{\#CL} (P_{gen_{(i)}} * P_{prop_{(i)}} * P_{logic_{(i)}} * \tau_{width_{(i)}} fs) \quad \text{CL} \right)$$

StartPoints and CL need to be captured by an EndPoint… hence data path derating factors exist.

Component Contribution to σ_{SEU} across Frequency and Gate Count

	Frequency	# of Gates in Path
EndPoint	Directly Proportional	N/A
StartPoint	Inversely Proportional	Inversely Proportional
CL	Directly Proportional	Directly Proportional

Current Use of NEPP FPGA Model

$$\sum_{k=1}^{\#EndPoint\ DFFs} \overset{\text{EndPoint Logic Masking}}{P_{logic(k)}} * \left(\begin{aligned} & \alpha P(fs)_{DFFSEU(k)} + \quad \text{EndPoint} \\ & \sum_{j=1}^{\#StartPoint\ DFFs} (\ \beta P(fs)_{DFFSEU(J)}(1-\tau_{dly(j)}fs)) * P_{logic(j)}) + \quad \text{StartPoints} \\ & \sum_{i=1}^{\#CL} (P_{gen_{(i)}} * P_{prop_{(i)}} * P_{logic_{(i)}} * \tau_{width_{(i)}}fs) \quad \text{CL} \end{aligned} \right)$$

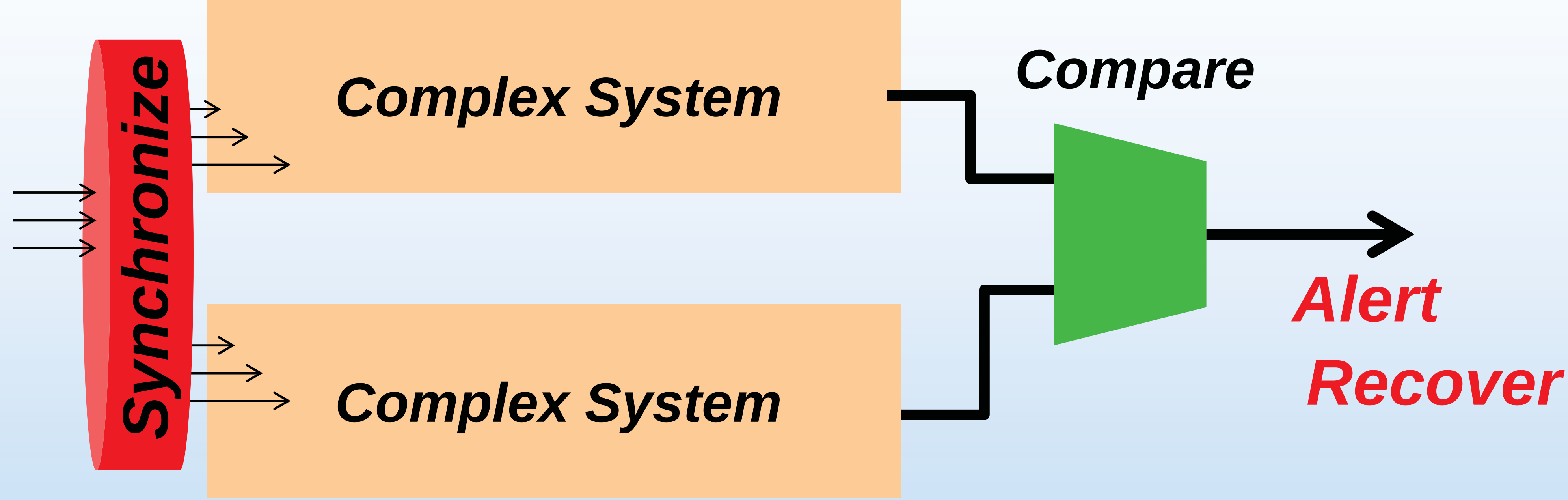
Currently, model is used to better understand heavy-ion SEU data:

Great for measuring mitigation scheme strength.

Fail-Safe Strategies

Dual Redundant Systems (Detection Systems)

Dual Redundancy Example



Synchronization is not always easy or predictable

- **Dual redundant systems cannot correct; they can only detect.**
- **Roll-back + dual redundancy is not a sufficient solution for systems with highly susceptible hardware.**
- **Alert systems must be highly reliable and verifiable.**

Mitigation – Fail Safe Strategies That Do Not Require Fault Detection but Provide SEU Masking and/or Correction:

Triple Modular Redundancy (TMR)

TMR Schemes Use Majority Voting

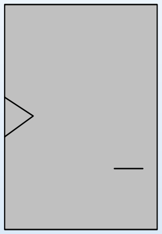
$$MajorityVoter = I1 \wedge I2 + I0 \wedge I2 + I0 \wedge I1$$

I0	I1	I2	Majority Voter
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
			1

Best 2 out of 3

Triplicate and Vote

Triplicate and Vote

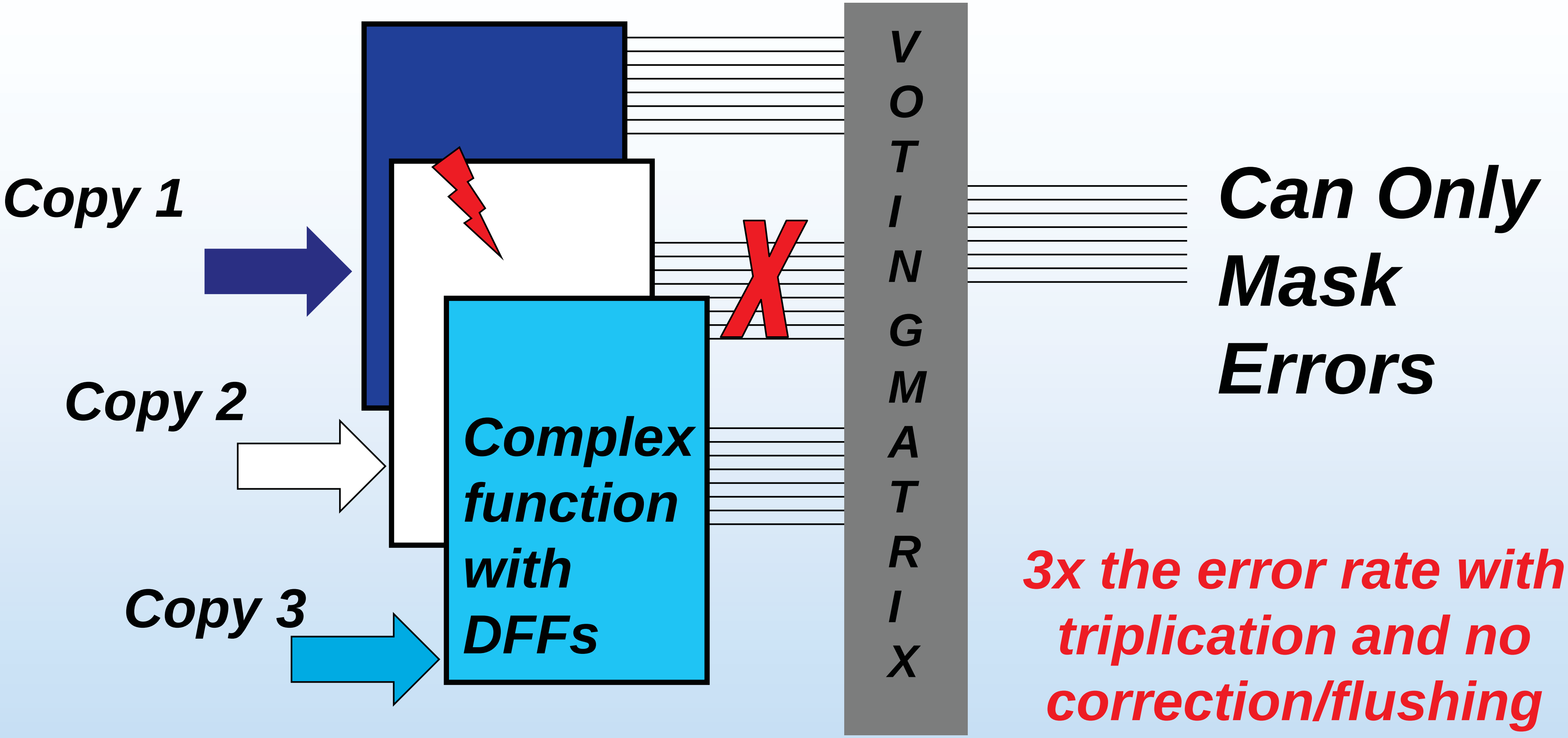


To be presented by Melanie Berg via WebEx at SERRESSA 2015 International School on the Effects of Radiation on Embedded Systems for Space Applications, Puebla, Mexico, November 30 to December 4, 2015.

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



3x the error rate with triplication and no correction/flushing

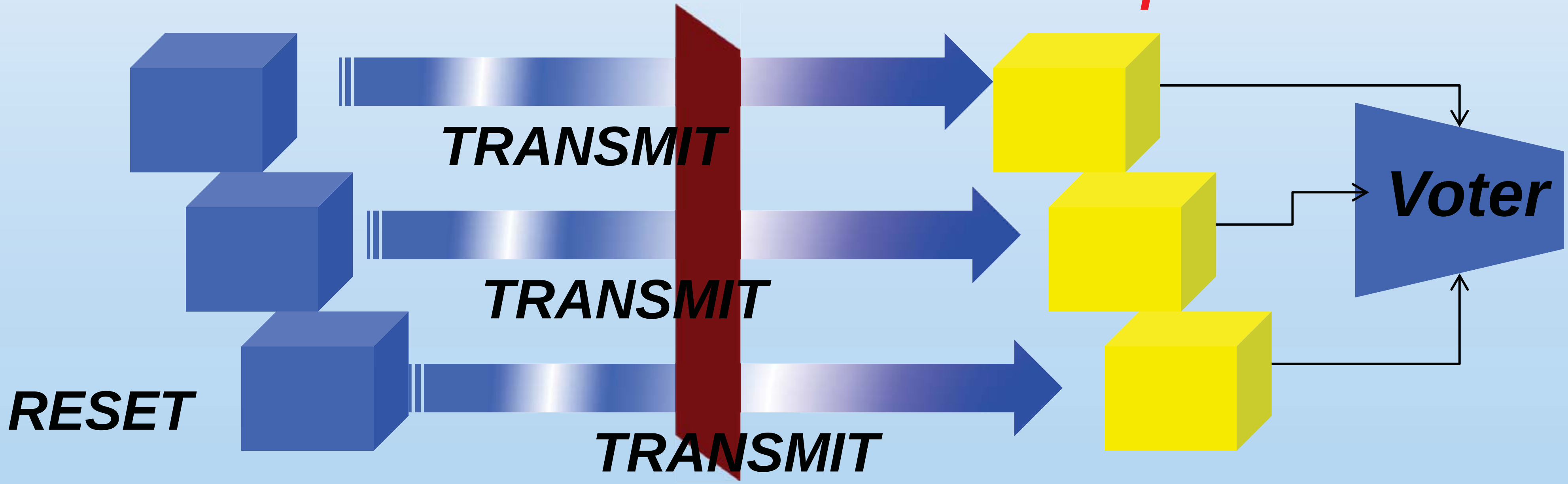
- Need Feedback or flushing to Correct
- Cannot apply internal correction from voted outputs
- If blocks are not regularly flushed (e.g. reset), Errors can accumulate – may not be an effective technique

To be presented by Melanie Berg via WebEx at SERRESSA 2015 International School on the Effects of Radiation on Embedded Systems for Space Applications, Puebla, Mexico, November 30 to December 4, 2015.

When BTMR is Beneficial: Examples of Flushable BTMR Designs

- **Shift Registers.**
- **Transmission channels: It is typical for transmission channels to send and reset after every sent packet.**
- **Lock-Step microprocessors that have relaxed requirements such that the microprocessors can be reset (or power-cycled) every so-often.**

Transmission channel example:

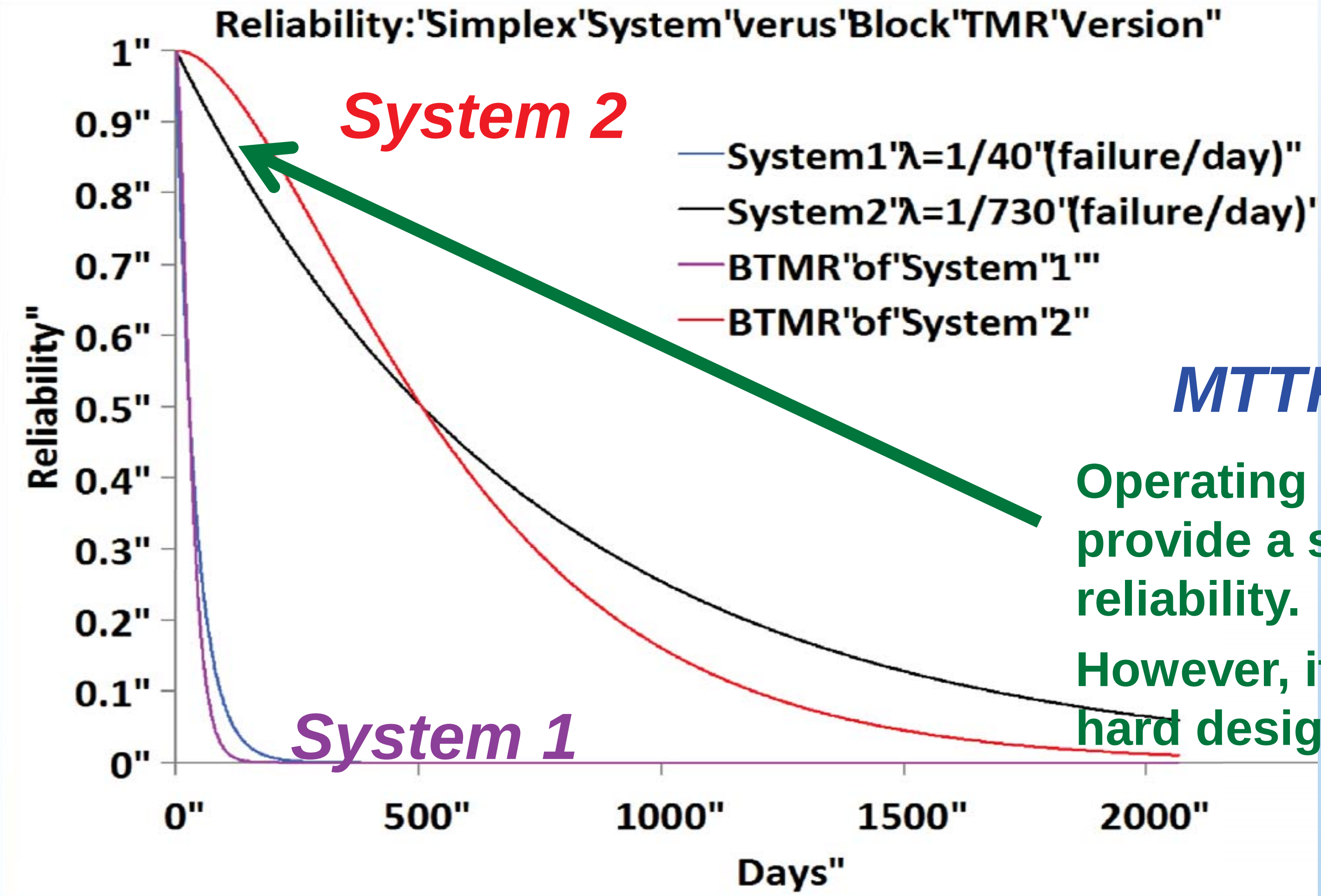


If The System Is Not Flushable, Then BTMR May Not Provide The Expected Level of Mitigation

- **BTMR can work well as a mitigation scheme if the expected MTTF >> expected window of correct operation.**
- **But… If the expected time to failure for one block is less than the required full-liveliness availability window, then BTMR doesn't buy you anything.**
- **If not thought out well, BTMR can actually be a detriment – complexity, power, and area, and false sense of performance.**

Explanation of BTMR Strength and Weakness using Classical Reliability Models

Relibility for 1 block (R_{block})	Relibility for BTMR (R_{BTMR})	Mean Time to Failure for 1 block ($MTTF_{block}$)	Mean Time to Failure BTMR ($MTTF_{BTMR}$)
$e^{-\lambda t}$	$3 e^{-2\lambda t} - 2 e^{-3\lambda t}$	$1/\lambda$	$(5/6\,\lambda) = 0.833/\lambda$



System 2

System 1

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

SEU Data

Overall:

$MTTF_{BTMR} < MTTF_{Block}$

Operating in this time interval will provide a slight increase in reliability.

However, it will provide a relatively hard design.

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 *DFF: Edge triggered flip-flop; CL: Combinatorial Logic*	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

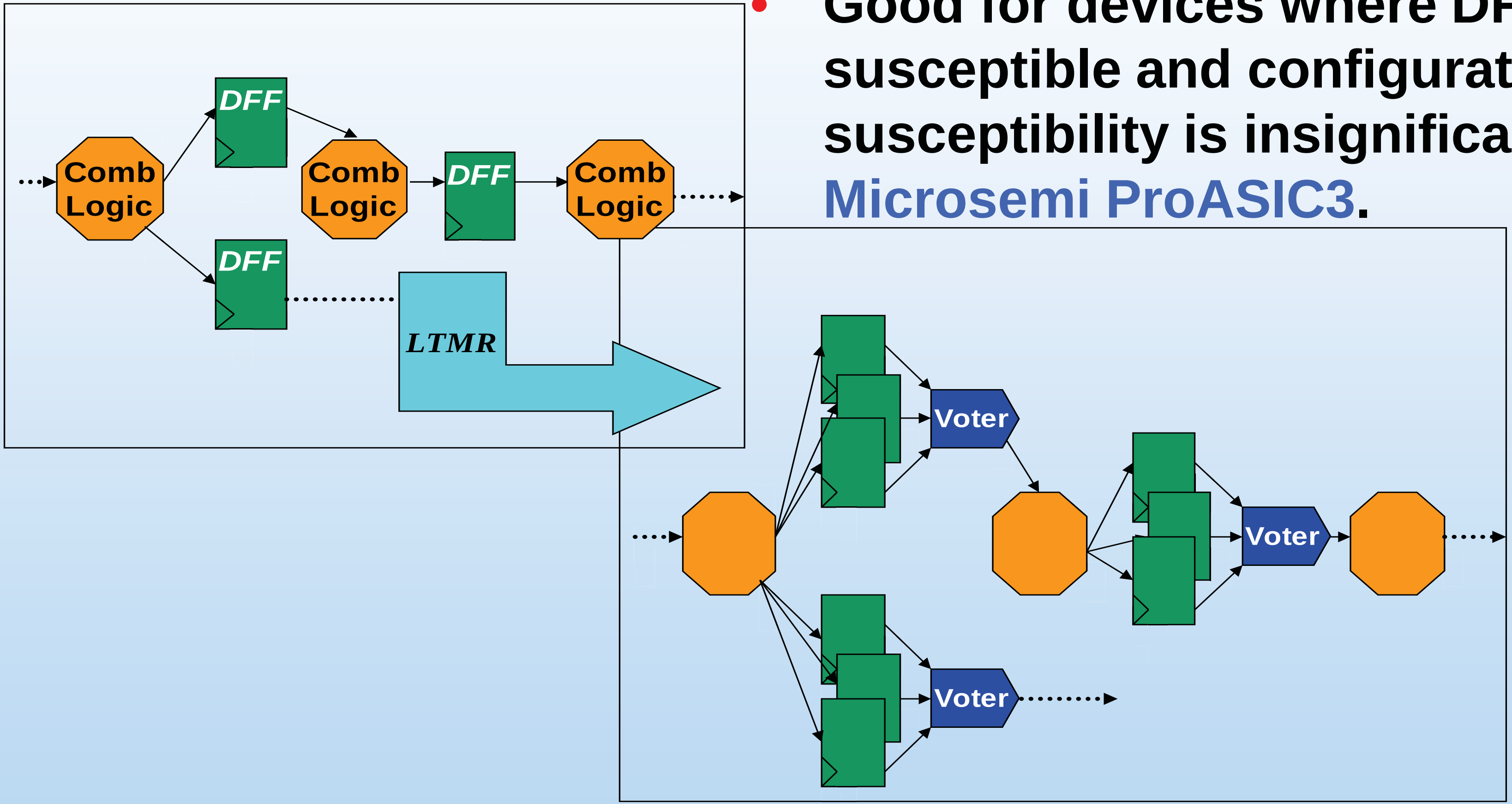
$$P(fs)_{error} \propto P_{configuration} + \underbrace{P(fs)_{functionalLogic}}_{P(fs)_{DFFSEU \rightarrow SEU} + P(fs)_{SET \rightarrow SEU}} + P_{SEFI}$$

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 ProASIC3.**

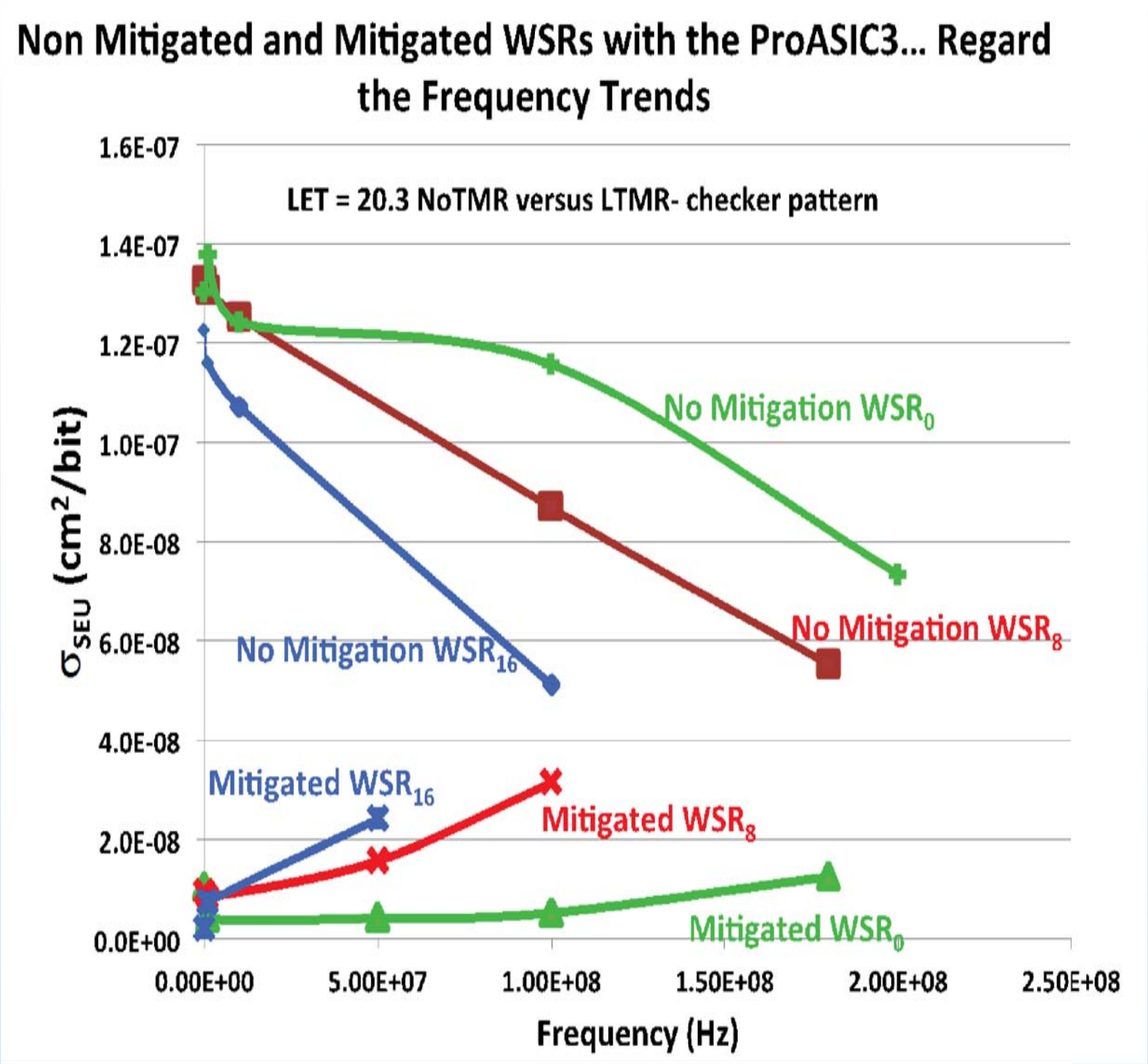


$$P(fs)_{error} \propto P_{configuration} + \underbrace{P(fs)_{functionalLogic}}_{\overset{0}{P(fs)_{DFFSEU \rightarrow SEU}} + P(fs)_{SET \rightarrow SEU}} + P_{SEFI}$$

Adding LTMR to a Microsemi ProASIC3 Device

LET: Linear Energy Transfer;
WSR: windowed shift register

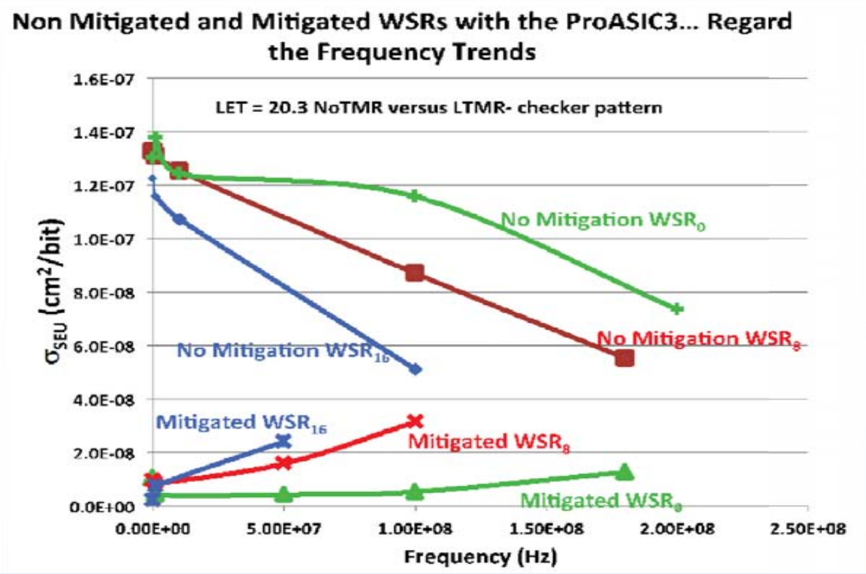
- **Microsemi ProASIC3 – DFFs are the most susceptible (to heavy-ion SEUs) data-path components.**
- **Adding LTMR decreases design sensitivity to SEUs.**

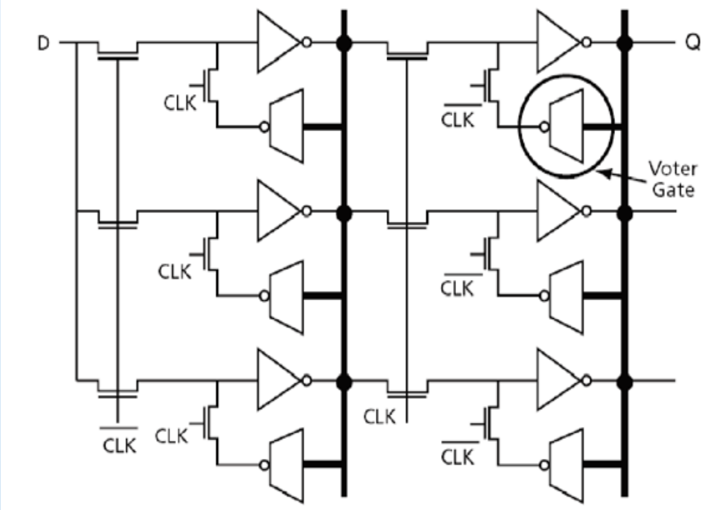


Adding LTMR to a Microsemi ProASIC3 Device versus RTAXs Embedded LTMR

LET: Linear Energy Transfer;
WSR: windowed shift register

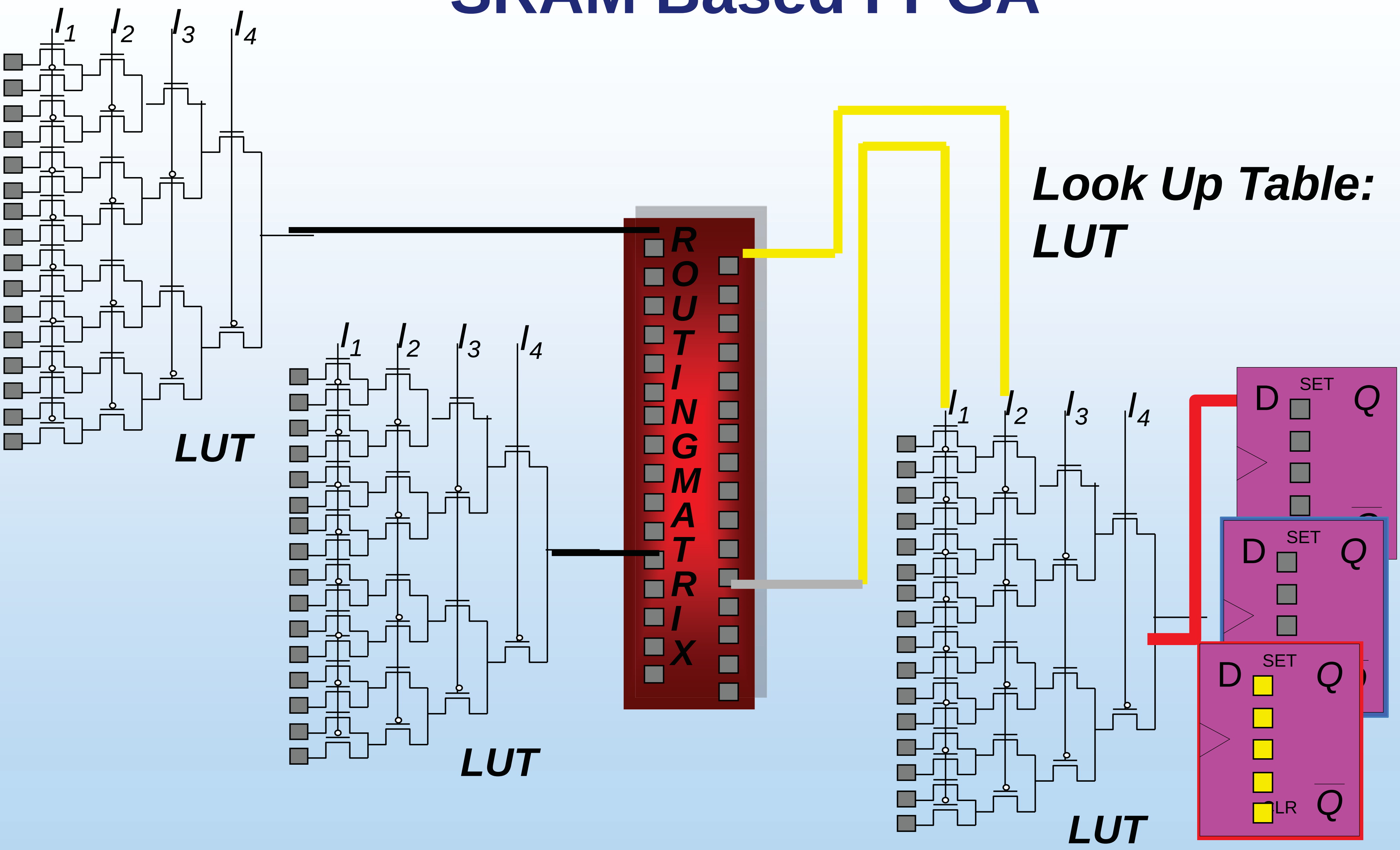
- **At lower LETs, user inserted LTMR to the ProASIC3 has similar SEU response to Microsemi RTAXs series.**
- **Higher LETs, clock tree upsets start to dominate and LTMR in the ProASIC3 is not as effective.**
- **For most critical applications, these cross-sections will produce acceptable upset rates.**





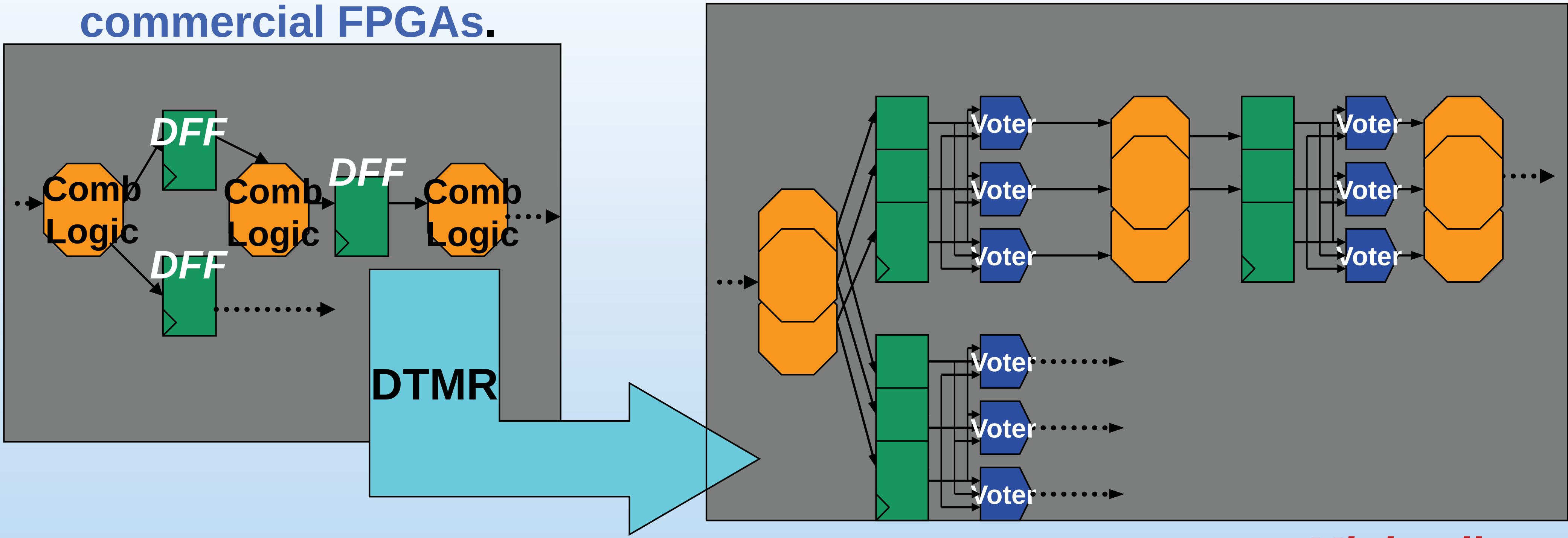
Embedded LTMR in a DFF cell RTAXs series.

LTMR Should Not Be Used in An SRAM Based FPGA



Distributed Triple Modular Redundancy (DTMR)

- Triple all data-paths and add voters after DFFs.
- 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.



$$P(f_s)_{error} \propto P_{configuration} + \underbrace{P(f_s)_{functionalLogic}}_{P(fs)_{DFFSEU \rightarrow SEU} + P(f_s)_{SET \rightarrow SEU}} + P_{SEFI}$$

Low

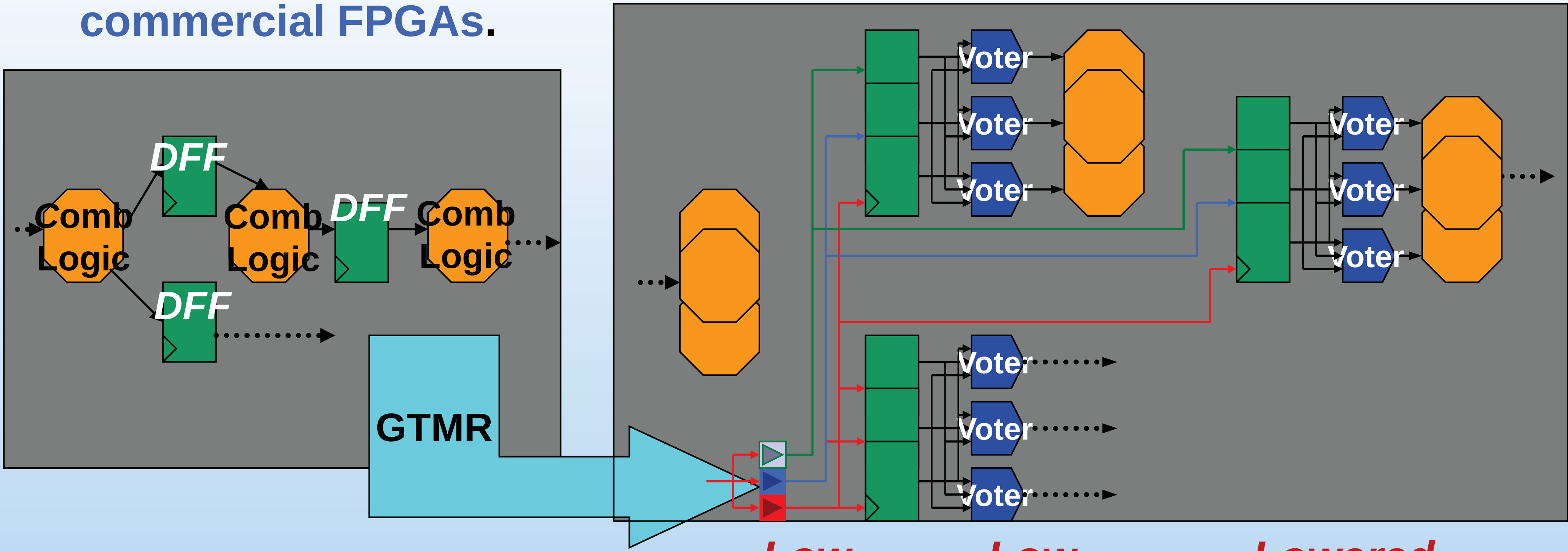
Minimally Lowered

0

Low

Global Triple Modular Redundancy (GTMR)

- Triple all clocks, data-paths and add voters after DFFs.
- GTMR has the same level of protection as DTMR; however, it also protects clock domains.
- Good for devices where configuration or DFFs + CL are more susceptible than project requirements; e.g., Xilinx and Altera commercial FPGAs.



$$P(f_s)_{error} \propto \overset{\text{Low}}{P_{configuration}} + \overset{\text{Low}}{\underbrace{P(f_s)_{functionalLogic}}_{\overset{\text{Low}}{P(fs)_{DFFSEU\to SEU}} + \overset{\text{Low}}{P(f_s)_{SET\to SEU}}}} + \overset{\text{Lowered}}{P_{SEFI}}$$

To be presented by Melanie Berg via WebEx at SERRESSA 2015 International School on the Effects of Radiation on Embedded Systems for Space Applications, Puebla, Mexico, November 30 to December 4, 2015.

Theoretically, GTMR Is The Strongest Mitigation Strategy… BUT…

- Triplicating a design and its global routes takes up a lot of power and area.
- Generally performed after synthesis by a tool– not part of RTL.
- Skew between clock domains must be minimized such that it is less than the feedback of a voter to its associated DFF:
  - Does the FPGA contain enough low skew clock trees? (each clock + its synchronized reset)x3.
  - Limit skew of clocks coming into the FPGA.
  - Limit skew of clocks from their input pin to their clock tree.
- Difficult to verify.

Currently, What Are The Biggest Challenges Regarding Mitigation Insertion?

- **Tool availability… Synopsys is now available.**
- **User's are not selecting the correct mitigation scheme for their target FPGA.**
- **Logic partitioning is not being performed when needed.**

FPGA Type	LTMR	DTMR	GTMR
Antifuse+LTMR: Microsemi RTAX or RTSX family	🟪	🟪 ????? 🟩	🟥
Commercial SRAM: Xilinx and Altera devices	🟥	🟩	🟪
Commercial Flash: Microsemi ProASIC family	🟩	🟪 ????? 🟩	🟥
Hardened SRAM: Xilinx V5QV	🟥	🟪 ????? 🟩	🟪

🟩 General Recommendation

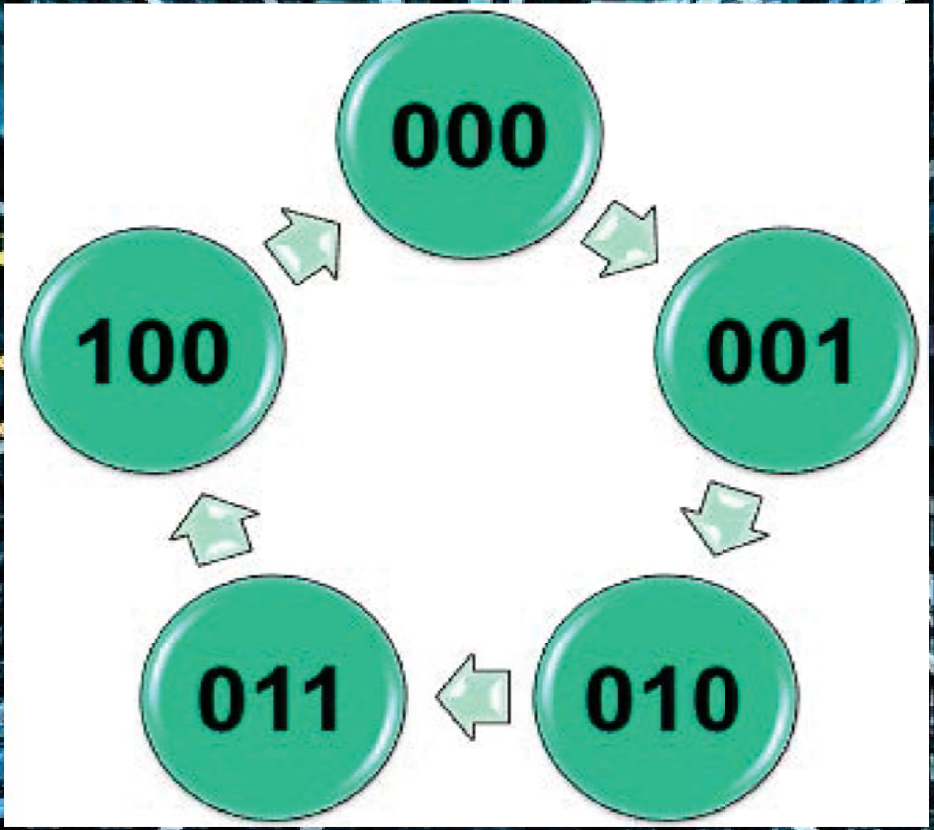
🟪 Not Recommended but may be a solution for some situations

🟥 Will not be a good solution

User versus Embedded Mitigation

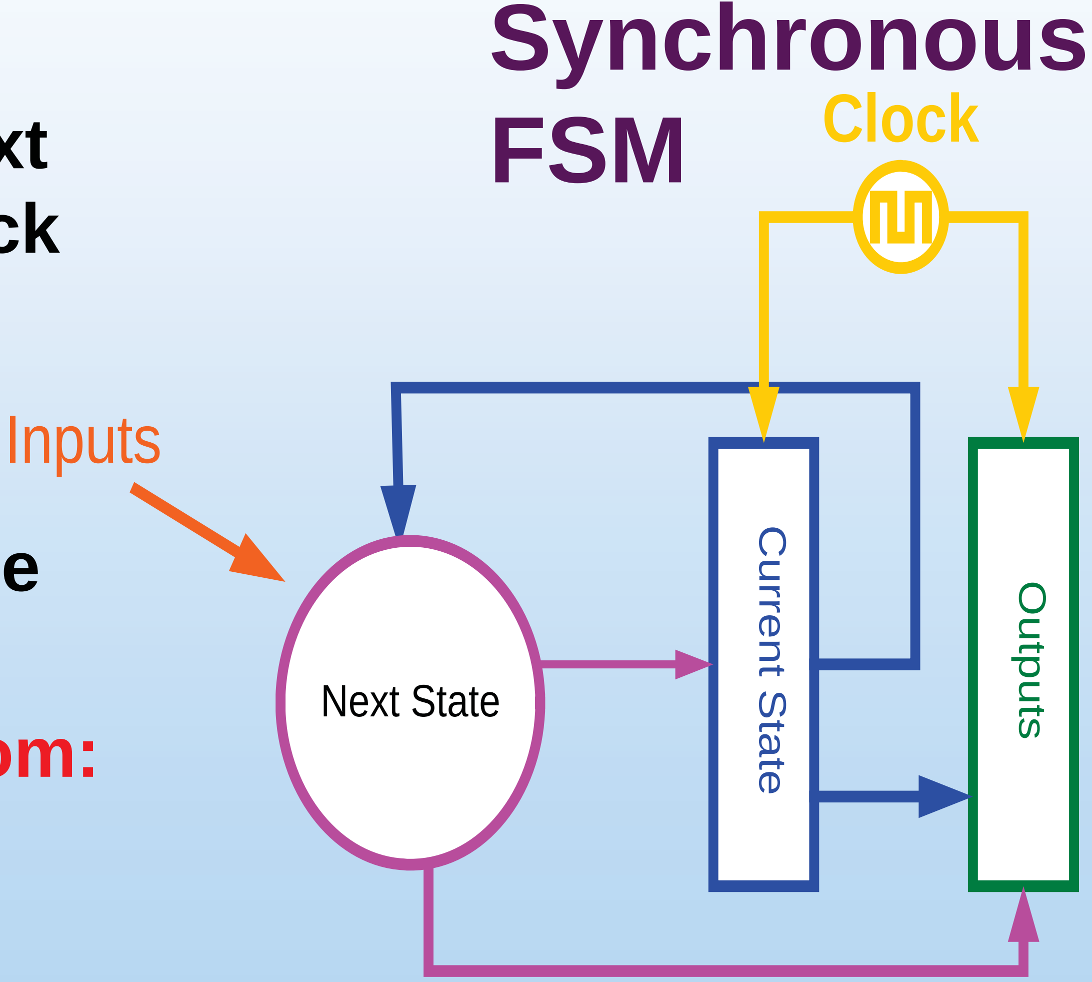
- **A subset of user inserted mitigation strategies have been presented.**
- **None of the strategies are 100% fail-safe.**
- **Depending on the project requirements, and the target device's SEU susceptibility, the most efficient mitigation strategy should be selected.**
- **In most cases, devices with embedded mitigation do not require additional (user inserted) mitigation.**

Fail-Safe State Machines

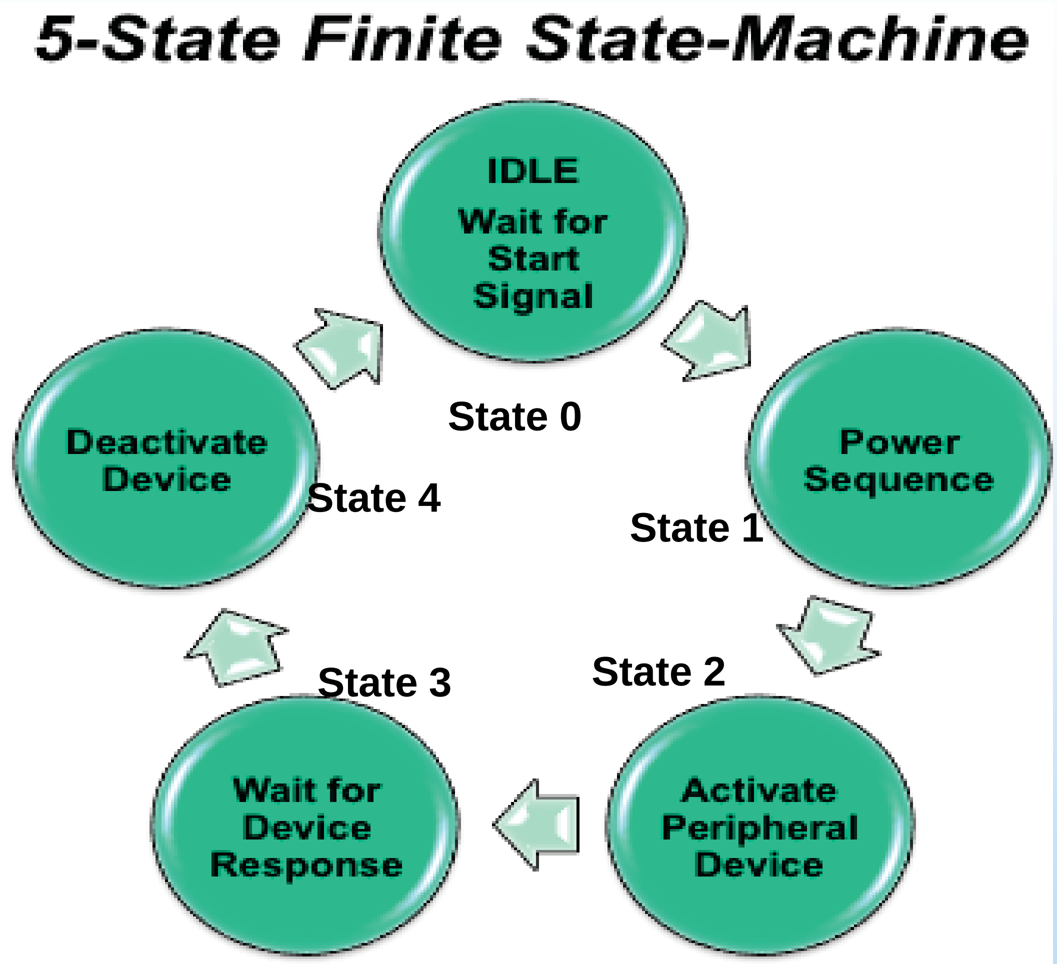


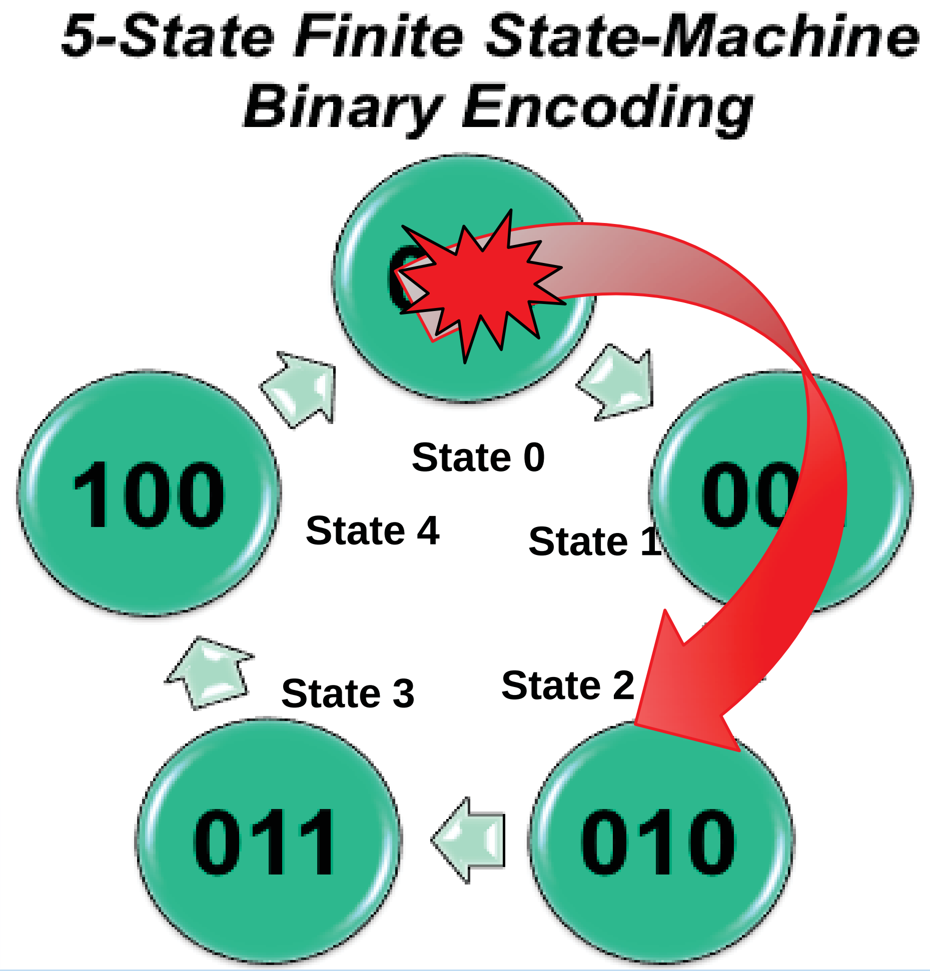
Synchronous FSMs and SEUs

- **A synchronous FSM is designed to deterministically transition through a pattern of defined states**
- **A synchronous FSM utilizes DFFs to hold its current state, transitions to a next state controlled by a clock edge and combinatorial logic, and only accepts inputs that have been synchronized to the same clock**
- **FSM SEUs can occur from:**
  - **Caught data-path SETs**
  - **DFF SEUs**
  - **Clock/Reset SETs**



5-State FSM Binary Encoding Example





Example of an FSM used to control a peripheral device

5-State FSM with each state encoded as binary numbers.

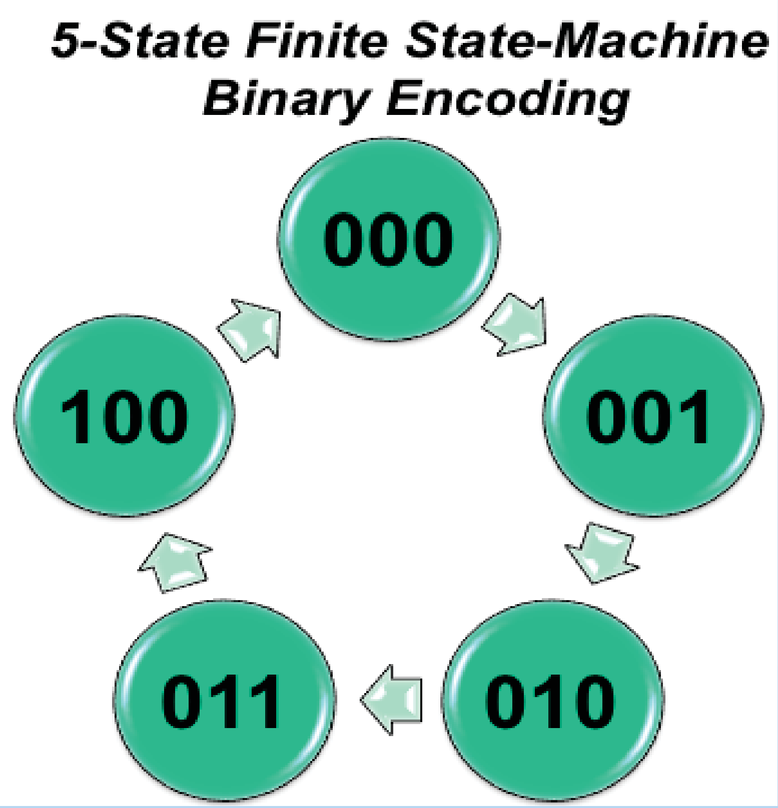
An SEU can change current state and cause a catastrophic event

How Do We Implement Fail-Safe FSMs?

- **Question: A designer states that all FSMs have been implemented as "safe", what do you expect?**
- **Correction? Detection? Masking?**
  - **What does correction mean?**
  - **All mitigation shall be defined unambiguously by the requirements and by the designer.**

Safe State Machines

- As currently defined by design tools and by some designers, the term “safe” state machine is a misnomer.
- Auto transitioning (“safe state-machine”) is a reaction to a small subset of incorrect transitions (unmapped states). They do not correct or mask (protect) against incorrect transitioning.

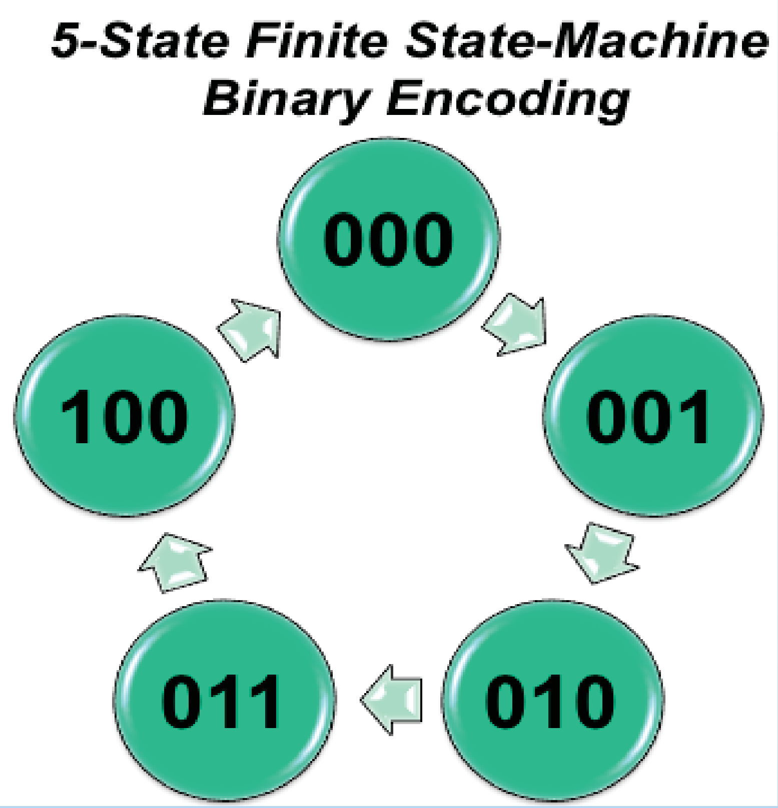


What happens if an SEU causes a transition from “001” to “101” ?

State	Mapped or Unmapped
000	Yes
001	Yes
010	Yes
011	Yes
100	Yes
101	No
110	No
111	No

Safe State Machines: What happens if an SEU causes a transition from “001” to “101” ?

- As currently implemented, a “safe” state machine will automatically transition to a reset (or “safe” state).
- Problem: this could be detrimental to your system

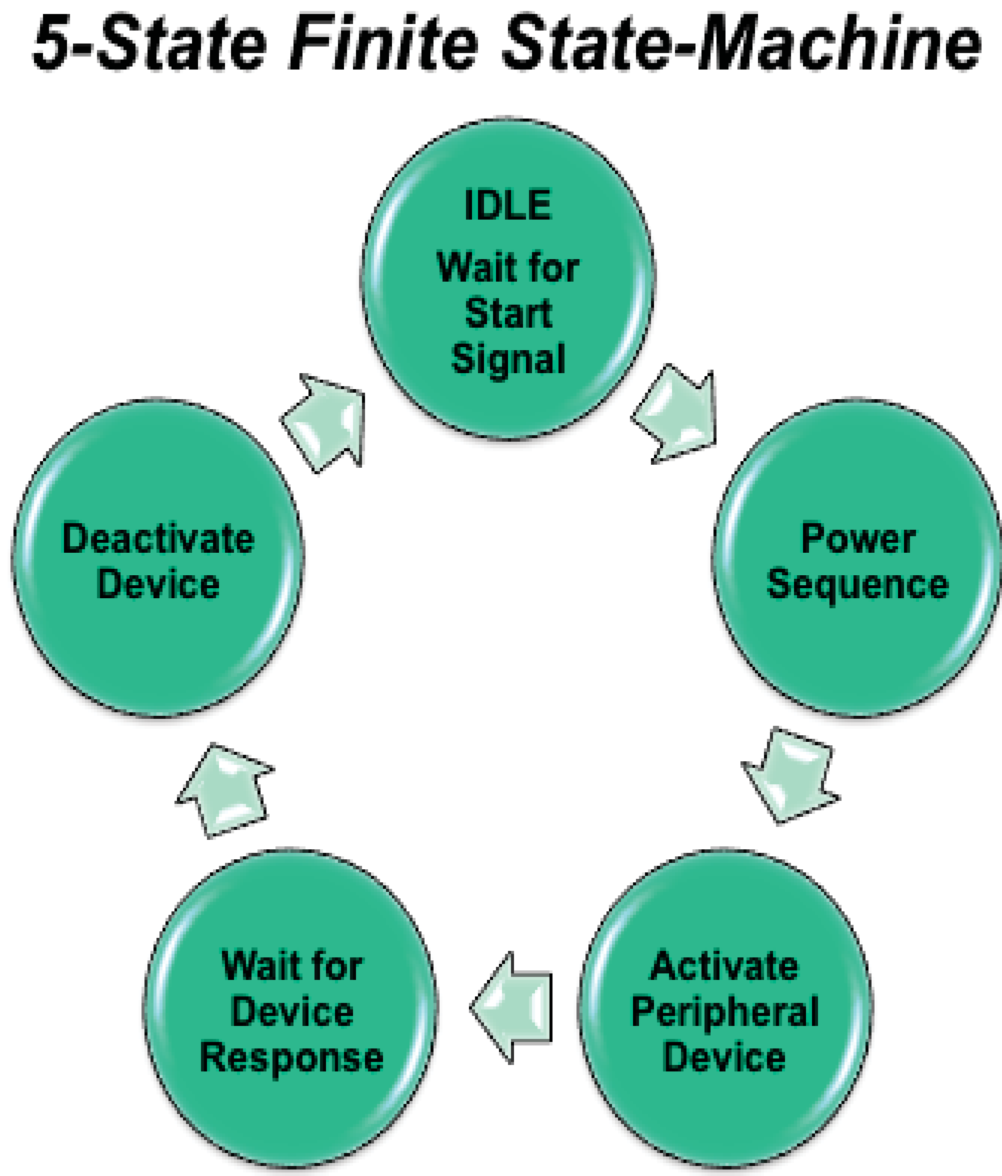


State	Mapped or Unmapped
000	Yes
001	Yes
010	Yes
011	Yes
100	Yes
101	No
110	No
111	No

To be presented by Melanie Berg via WebEx at SERRESSA 2015 International School on the Effects of Radiation on Embedded Systems for Space Applications, Puebla, Mexico, November 30 to December 4, 2015.

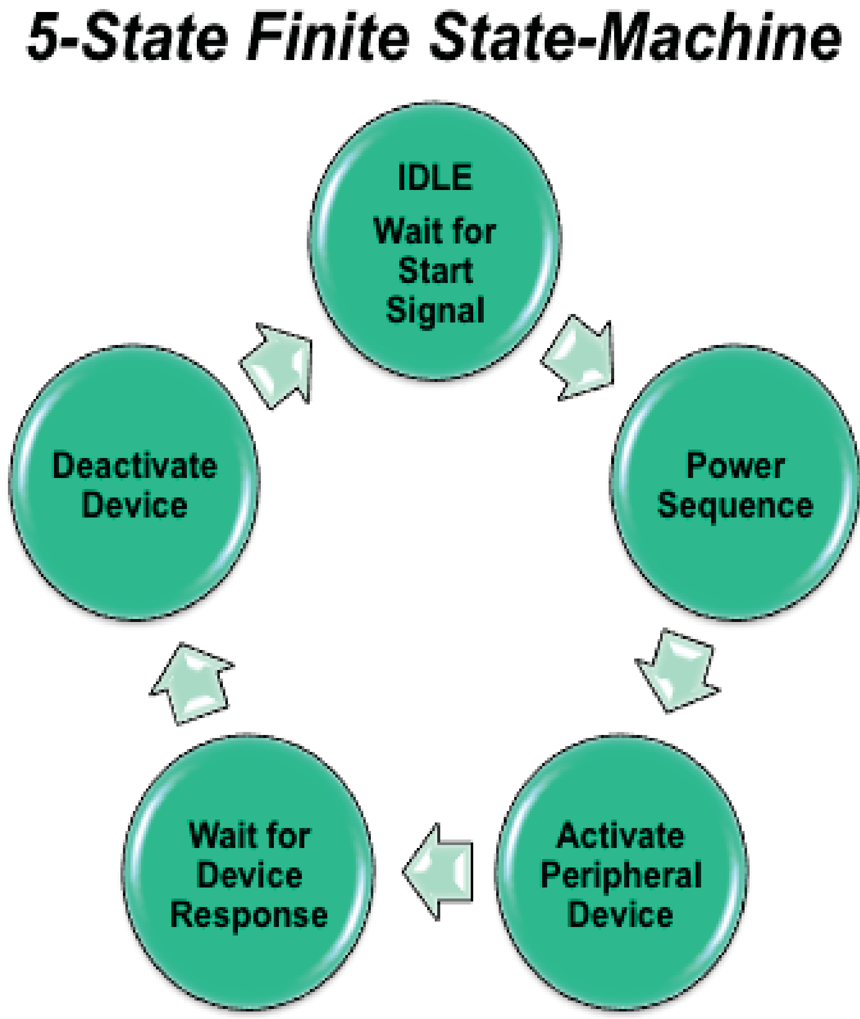
Problems with Current "Safe" FSM Definition

- Sounds more safe than what it really is.
- Does not do anything for incorrect transitions into mapped states.
- Does not correct the state:
  - Something that is supposed to be on will abruptly shut off.
  - Other FSMs or control logic can become unsynchronized with the bad FSM; with or without the automated jump to a "safe" state.


5-State Finite State-Machine

Can Auto-transitioning Work for Your Mission?

- **Auto-transitioning can work if incorrect sequencing of your FSM will not cause system failure; e.g. mathematical logic control.**
- **Auto-transitioning can be acceptable if it is used in conjunction with a detection flag. The detection flag must propagate to all necessary logic.**
- **But remember, there is no protection or detection with auto-transitioning when incorrectly transitioning to a mapped state.**



Auto-transitioning + detection is available with computer aided design (CAD) tools.

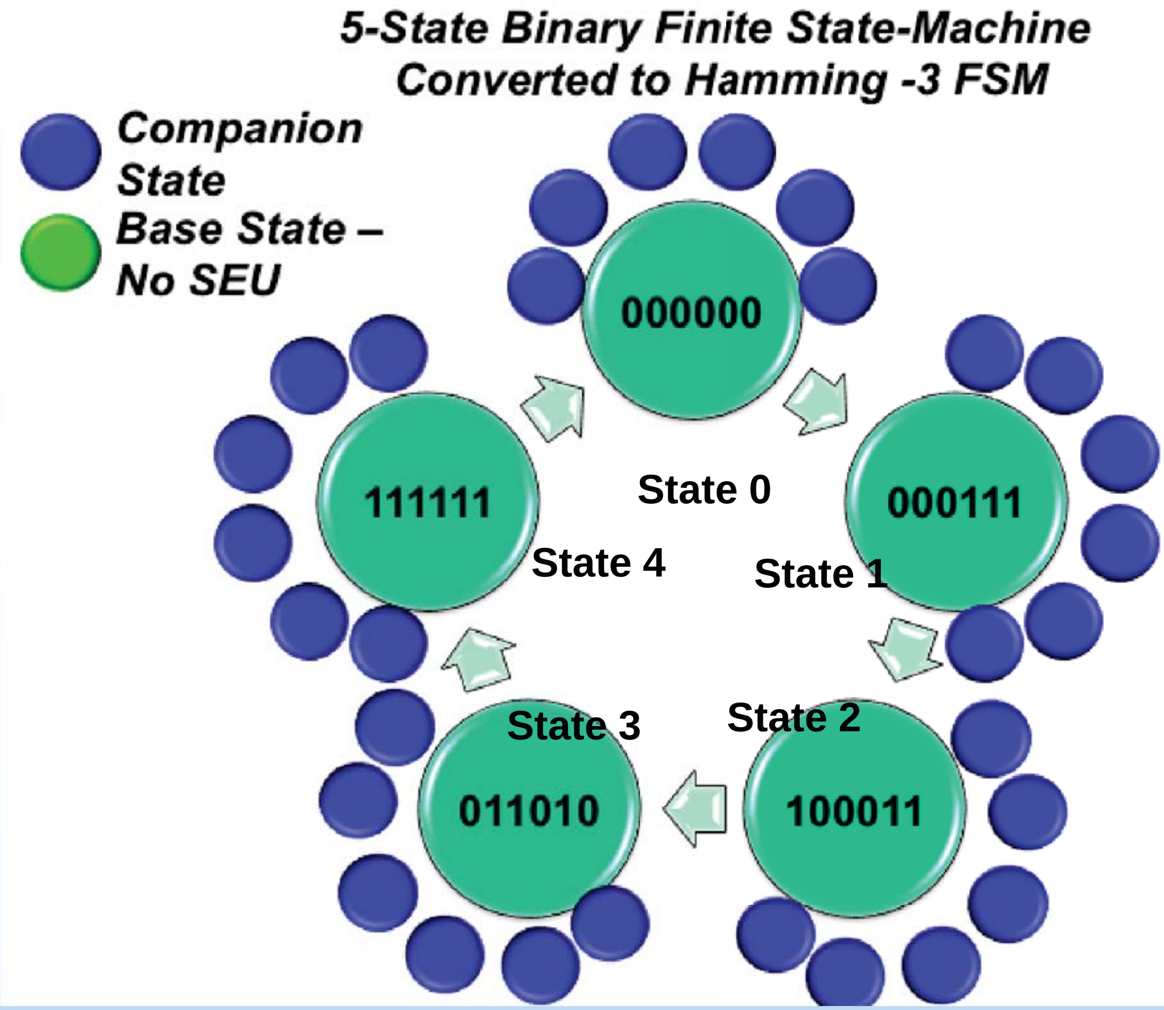
Implementing Corrective Logic for FSMs

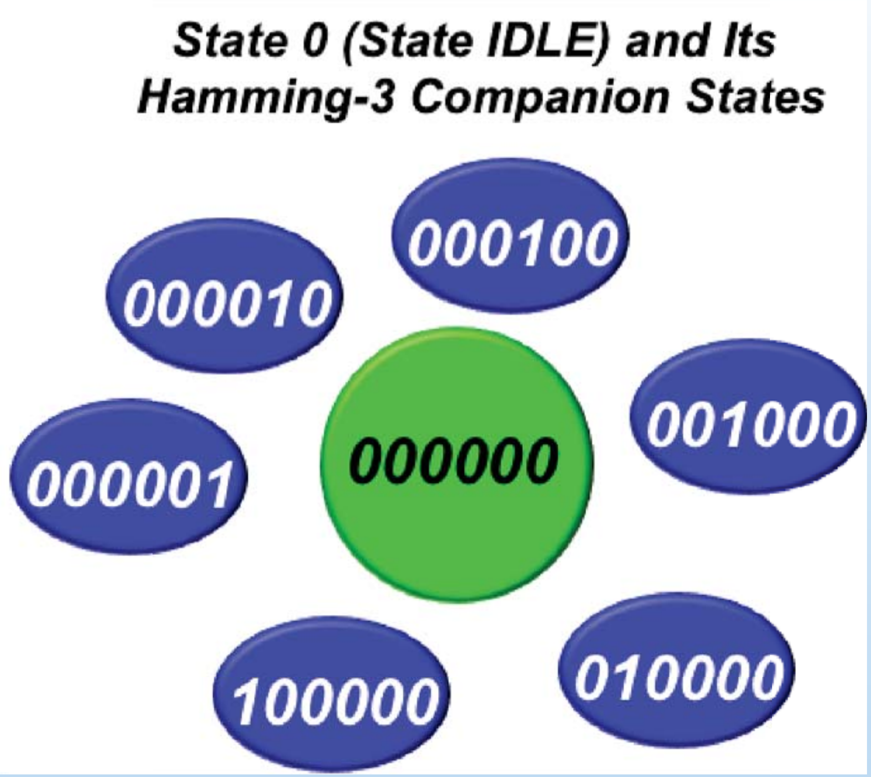
- **FPGAs with hardened configuration:**
  - **LTMR: Triplicate each DFF and use a majority voter.**
    - **The triplication + voter is treated as one DFF**
    - **Encoding doesn't change**
    - **Resultant FSM has 3 times the number of DFFs than the original encoding scheme.**
    - **Combinatorial logic (not including the voters) does not change**
  - **Hamming Code-3: requires a new encoding scheme.**
- **FPGAs with commercial SRAM configuration: DTMR is suggested.**

There are computer aided design tools (CAD) that can assist in adding all of the above mitigation strategies.

FSM Fault Tolerance:

5-State Conversion to a Hamming Code-3 FSM

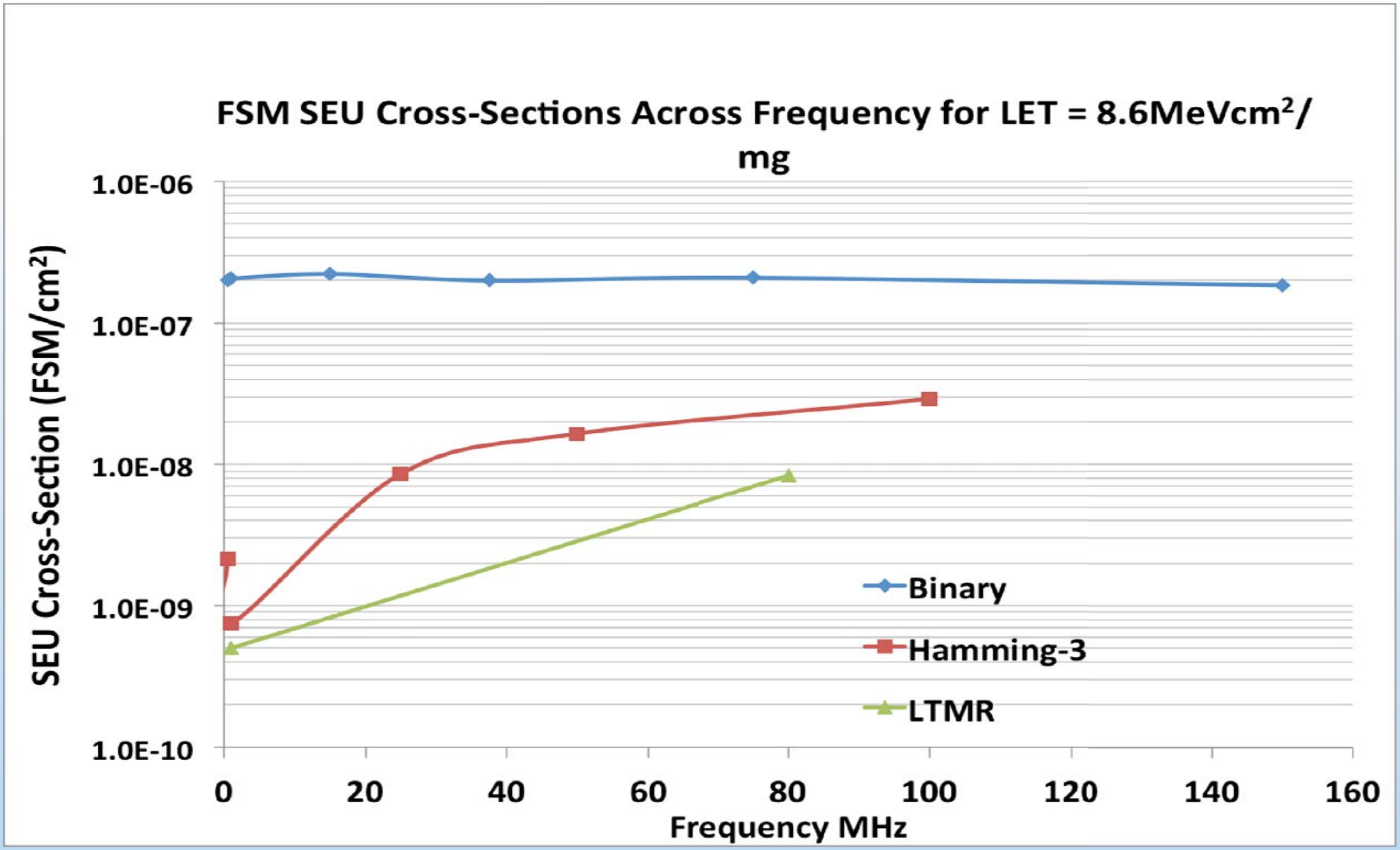




Hamming Code-3 FSM Diagram for a 5 Base-State FSM: Would need 5*7=35 FSM states to be represented… 6 DFFs

A closer look at a base-state (state 0) and its companion-states

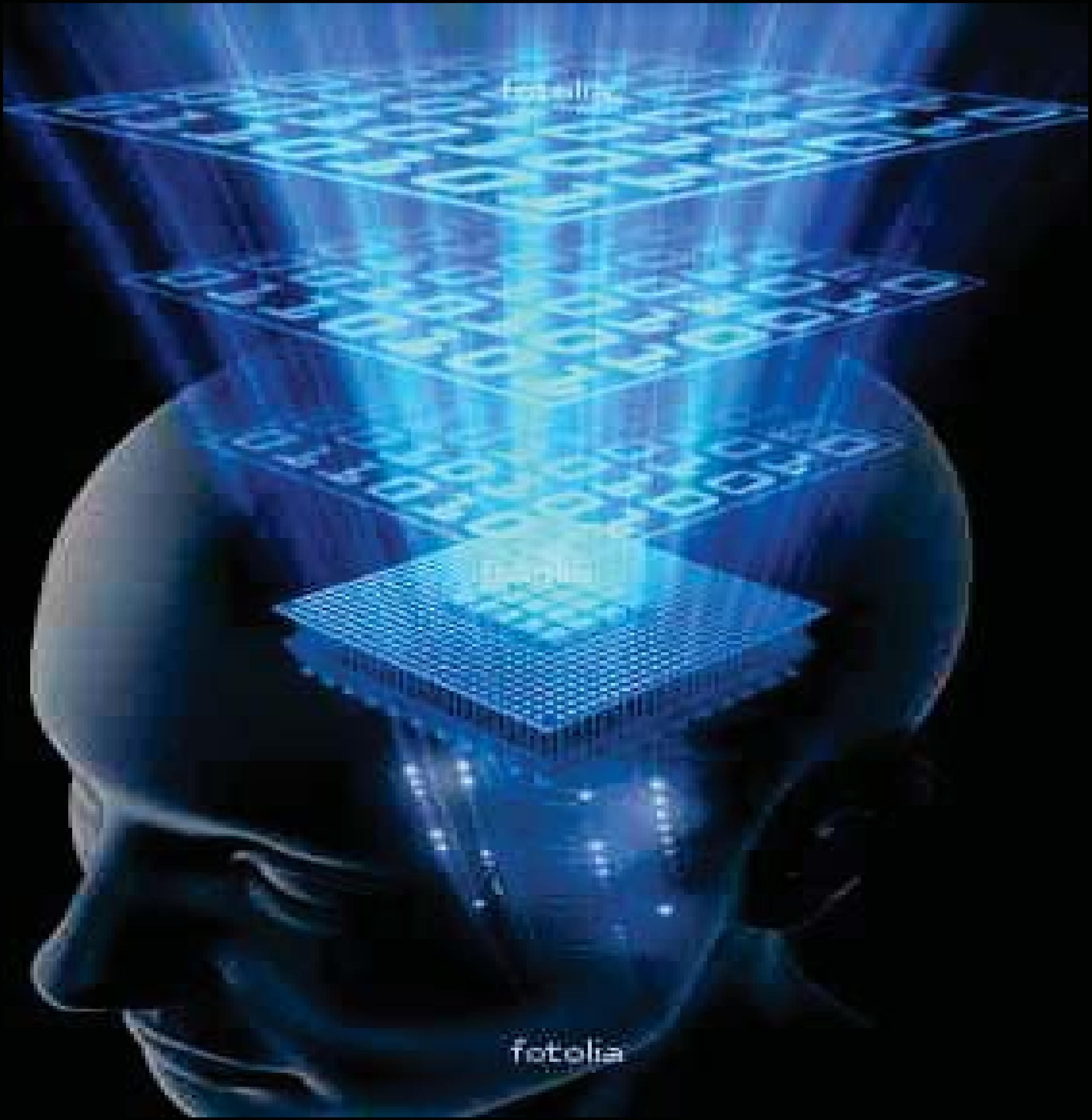
ProASIC3 Heavy-Ion FSM SEU Testing



SEU cross-sections per FSM.

Scale is Log-Linear

Some Thoughts



Concerns and Challenges of Today and Tomorrow for Mitigation Insertion

- User insertion of mitigation strategies in most FPGA 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 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.
- Embedded mitigation has helped in the design process. However, it is proving to be an ever-increasing challenge for manufacturers.
  - We (users) want embedded systems: cheaper, faster, and less power hungry.
  - However, heritage has proven that for critical applications, embedded systems have provided excellent performance and reliability.

Summary

- **For critical applications, mitigation may be required.**
- **Determine the correct mitigation scheme for your mission while incorporating given requirements:**
  - **Understand the susceptibility of the target FPGA and how it responds to other devices.**
  - **Investigate if the selected mitigation strategy is compatible to the target FPGA.**
  - **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.**