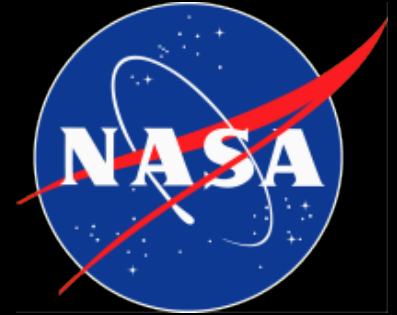


Unclassified



# Reliable Design Versus Trust

**Melanie Berg**

**AS&D in support of NASA/GSFC**

**[Melanie.D.Berg@NASA.gov](mailto:Melanie.D.Berg@NASA.gov)**

**Kenneth A. LaBel**

**[ken.label@nasa.gov](mailto:ken.label@nasa.gov)**



# Acronyms

Acronym	Definition
ASIC	Application specific integrated circuit (ASIC)
BFBMs	Bus functional Models (BFBMs)
BRAM	Block random access memory (BRAM)
CLB	Configurable Logic Block (CLB)
CM	Configuration Management (CM)
CRCs	Cyclic redundancy codes (CRCs)
DFR	Design for Reliability (DFR)
DFT	Design for Test (DFT)
DFV	Design for Verification (DFV)
DSP	Digital Signal Processing (DSP)
EDF	Evolutionary Digital Filter (EDF)
EDIF	Electronic Design Interchange Format (EDIF)
FPGA	Field programmable gate array (FPGA)
GNL	Gate Level Netlist (GLN)
GR	Global Route (GR)
HDL	Hardware Design Language (HDL)
I/O	Input – output (I/O)
IP	Intellectual Property (IP)
NASA	National Aeronautics and Space Administration (NASA)
NEPP	NASA Electronic Parts and Packaging (NEPP) Program
PR	Place and Route (PR)
R	Reliability (R)
SOC	System on a chip (SOC)
SRAM	Static random access memory (SRAM)

- **Independent caches organized as a hierarchy (L1, L2, etc.) (L2 Cache)**
- **FPGAs use JTAG to provide access to their programming debug/emulation functions (JTAG)**

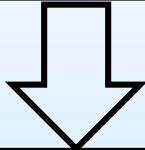
# High-Level Field Programmable Gate Array (FPGA) Design Flow:



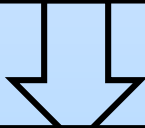
## From The Manufacturer To System Insertion

*HDL: Hardware Design Language*

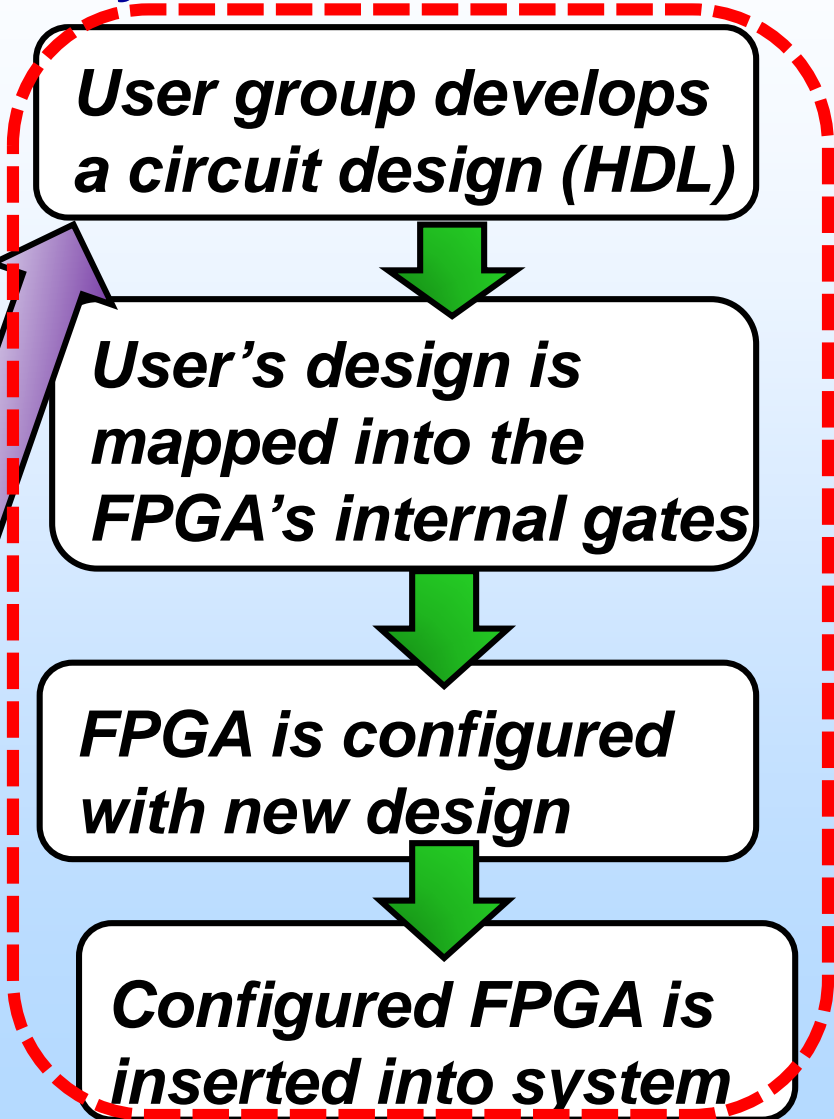
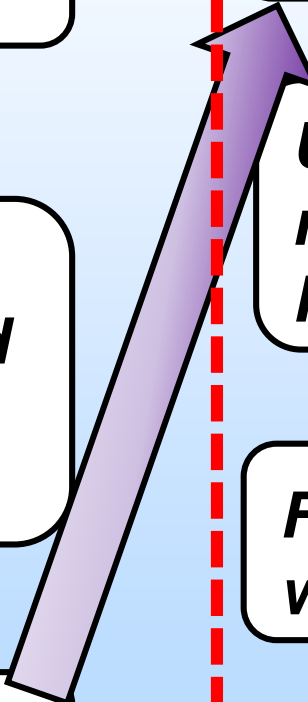
**Manufacturer develops an FPGA Architecture**



**Mask is provided to Fabrication Foundry and the FPGA device is created**



**User organization procures FPGA devices**



**User group develops a circuit design (HDL)**



**User's design is mapped into the FPGA's internal gates**



**FPGA is configured with new design**



**Configured FPGA is inserted into system**

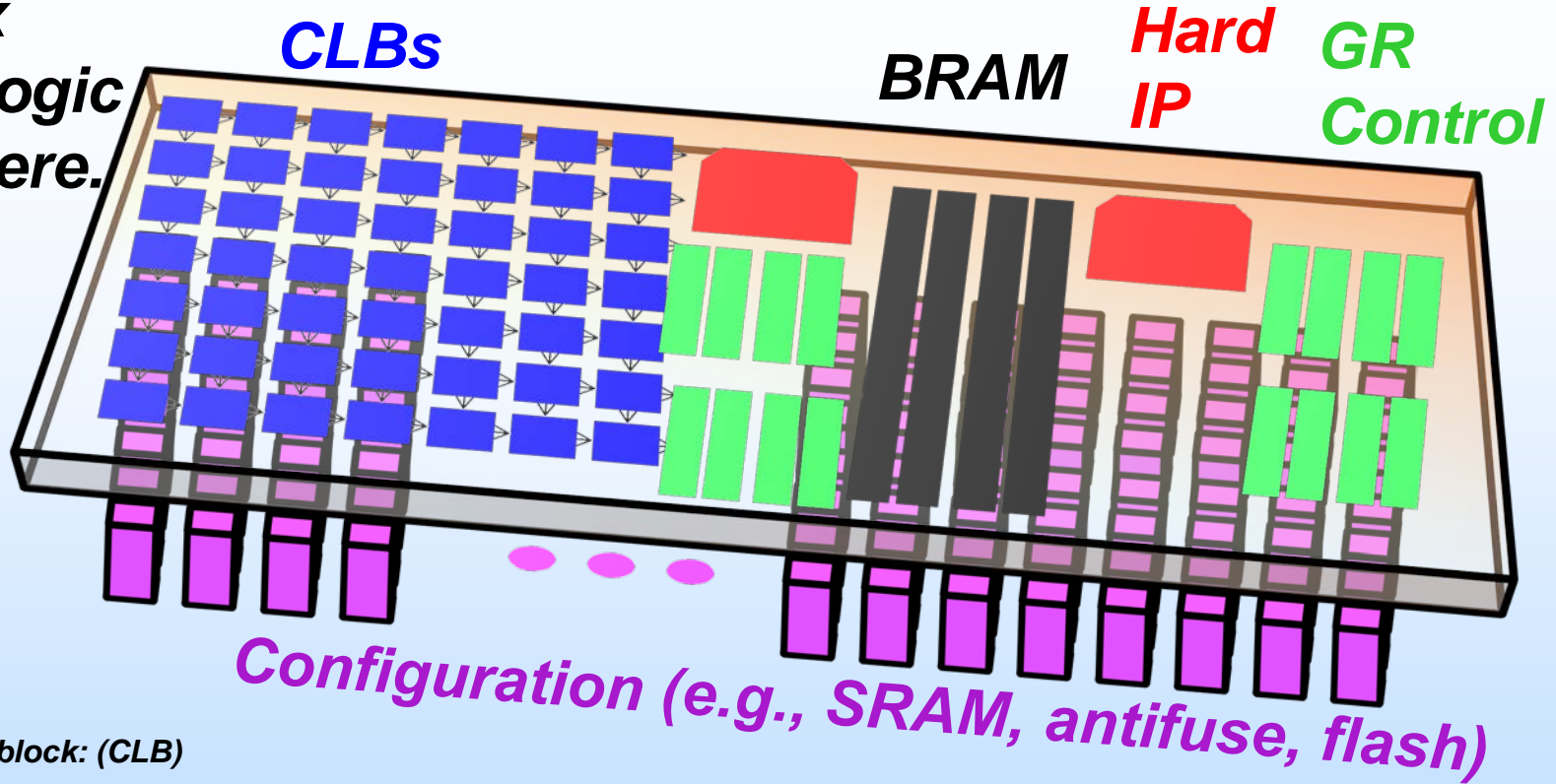


# Scope of Presentation.

- **This presentation focuses on reliability and trust for the user's portion of the FPGA design flow.**
- **It is assumed that FPGA internal components (configuration cells, routing, logic cells, hard intellectual property (IP), global routes, protection mechanisms, etc.) are tested by the manufacturer prior to hand-off to the user.**
- **The objective is to present the challenges of creating reliable and trusted designs.**
  - **What makes a design vulnerable to functional flaws (reliability) or attackers (trust)?**
  - **What are the challenges for verifying a reliable design versus a trusted design?**

# FPGA Reliable Operation

Complex routing logic everywhere.



Configurable logic block: (CLB)

Block random access memory: (BRAM)

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

Global Routes: (GR)

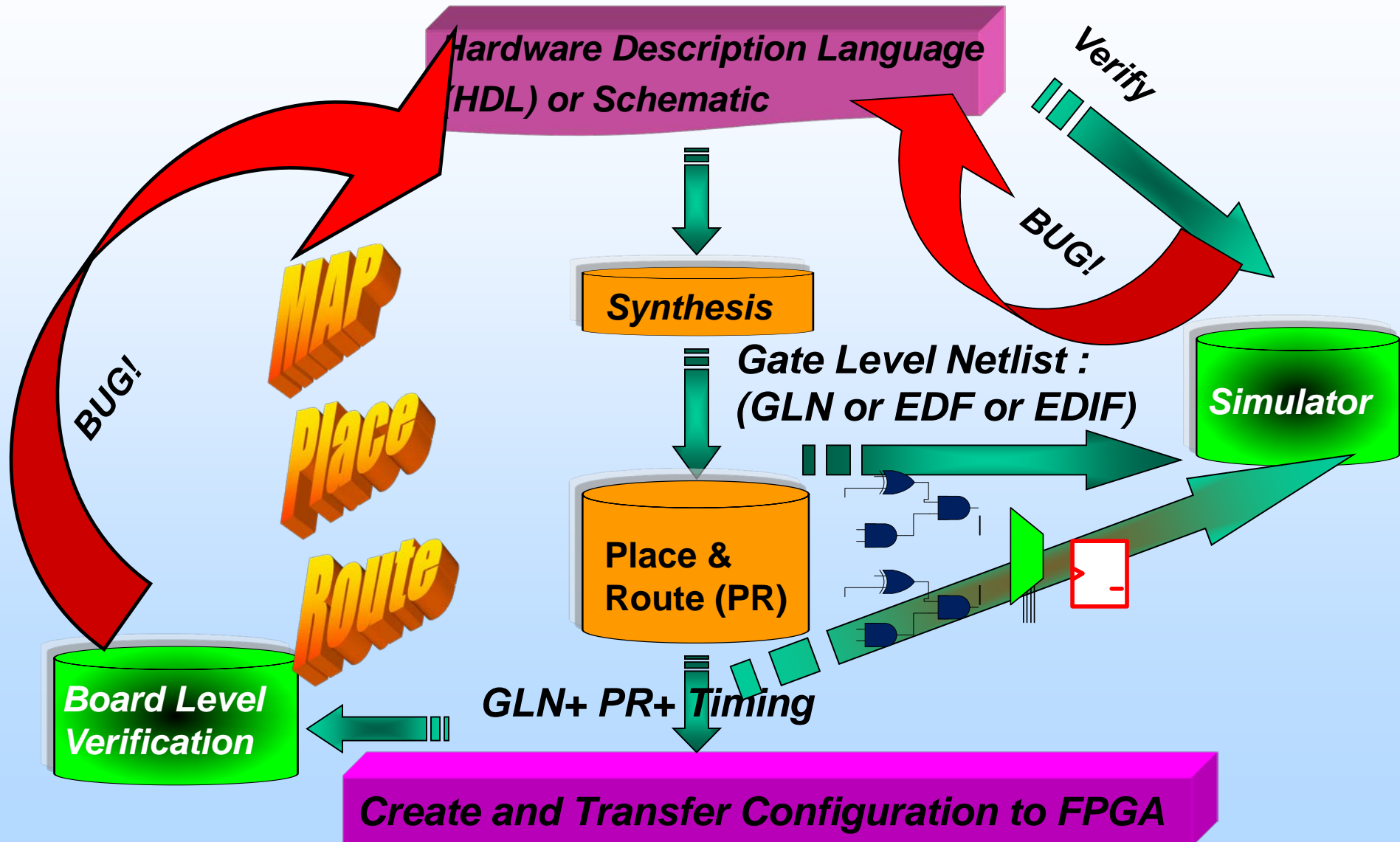
Reliability:  $R$

$$R_{operation} \propto R_{Configuration} + R_{CLB} + R_{BRAM} + R_{GL} + R_{IP} + R_{HiddenLogic} + R_{parametrics}$$

**Reliable operation depends on a variety of parameters.**



# A Closer Look at The FPGA Design Process from The User's Perspective



# Functional Design Development



- **Both the design team and tool-set are expected to be reliable and trusted.**
- **Contractors are selected by a secure organization... but ... the design team is selected by the contractor.**
  - **How well do you trust the members of the design team?**
  - **How many levels of contracting exist?**
  - **Are the designers trained properly with pertinent design and verification experience?**
  - **Are there protection mechanisms for possible inside attacks?**
- **Tools are selected by the design team.**
  - **Are the tools from an accredited design organization?**
  - **Is there a stipulation in the contract that the design team is required to use trusted tools?**
- **Is the contractor's full design flow visible and are there the appropriate checks and balances in place (documentation and reviews)?**



**Any weakness within your design flow:  
personnel, design methodology,  
design tools, verification process, and  
check & balances  
... leaves an open door for an inside  
attack.**

***We need to focus on the areas that are  
most vulnerable.***



# Synthesis Tools



- User can select a synthesis tool from a trusted 3<sup>rd</sup> party (e.g., Synopsys) or from the FPGA's manufacturer.
- It is the synthesis tool's responsibility to:
  - Interpret/analyze/optimize the user's HDL, and
  - select component cells from the FPGA device's cell library to create the described hardware functionality.
- **Vulnerability:** It is difficult to verify that the expected gate-level output matches the intended HDL.
- Bad synthesis output can be due to:
  - Poor user written HDL,
  - Mediocre synthesis tools, or
  - Malicious synthesis tools.
- **Vulnerability example:** did the synthesis tool optimize away necessary logic? Can you detect that the necessary logic does not exist?
- Best tool available for synthesis output verification: equivalence checking. **Doesn't work for all trust cases.**

# Place and Route (PR) and Configuration Management (CM) Tools: Vulnerabilities



- Configuration contains all of the mapped place and route information.
- PR and CM from tools provided by the manufacturer.
- Does the manufacturer have a trusted tool group?
  - Offshore designers, Software IP, or University contributions.
- **Vulnerability Example (1)**: Objects can be “optimized” away (or erased) during PR or CM.
- **Vulnerability Example (2)**: Configuration can be changed to disrupt function, timing, signal integrity, area, or power requirements.

# PR and CM Tools: Current Solutions



- PR and CM tools produced by major manufacturers are not highly vulnerable because of their widespread usage. Bugs or incorrect products are easier to detect.
- Functional verification is performed at the system level.
  - Can find many PR and CM bugs at the system level.
  - **Challenge:** it can be difficult to find corner case bugs.
- Tools are available to perform a form of equivalence checking and formal verification to help verify that all logic exists.
  - It is important to note that the tools are limited in their success.
  - **Challenges:** size of circuit and redundancy.
- Cyclic redundancy codes (CRCs) and Keys are used to identify unique configurations.
  - They are usually checked in design reviews.
  - **Challenge:** Due to last minute design changes, it is rare that the last configuration downloaded to the FPGA has been reviewed (lack of check and balances).

# Design Methodology and Reliable Operation Considerations



*HDL: hardware description language*

**Number of Clock Domains**

**Clock Balancing**

**Reset Structure**

**Area**

**Metastability**

**Power (Hot-spots)**

**Long Traces  
(charge sharing)**

**I/O Standard Selection**

**Creation of Latches versus Edge-triggered flip-flops**

**I/O Rings and Pin Switching  
(ground-bounce)**

**Static Timing Analysis ...  
Setup/hold time violations (race conditions)**

**Synthesis tool interpretation of HDL**

**These concerns are FPGA/ASIC hardware design specific.  
They are not Software and Firmware design concerns.**

# Vulnerabilities when Assuming FPGA is Software or Firmware



- **Reliability:** If a design is not managed as a hardware solution (i.e., if a design is managed as a software or firmware product), then reliability will be compromised.
- **Trust:**
  - Overlooked hardware considerations will leave vulnerabilities for attackers.
  - In other words, if an attacker knows a design team is not following proper design techniques, bugs can be easily inserted.
- **Reliability and trust:** A strong verification process might be able to find most bugs. However, with highly complex designs, verification (as performed in the FPGA design world) is not always sufficient.

# Difference between Reliability and Trust



- **Types of functional reliability failures are fairly well known:**
  - **Bad design practices: lack of training, asynchronous design, signal integrity, incorrect clock domain crossings, metastability, set-up time violations, race conditions, incorrect logic implementation, ambiguous documentation.**
  - **Bad verification practices.**
  - **Misinterpreting FPGA internal components.**
  - **Misinterpreting HDL versus software or firmware.**
- **Types of trust failures are not as well known because there's always an attacker searching for vulnerabilities.**
- **It's difficult to tell the difference between bad design practice and malicious intent.**

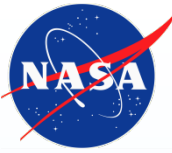
# Why is Reliable Design Practice Important with Regards to Trust



- **Attackers will try to take advantage of vulnerabilities.**
- **Bad design is a vulnerability.**
- **Due to the complexity of modern day designs it is difficult to find bugs in a design.**
  - **Limitations in verification exist.**
  - **Designs are becoming extremely complex.**
  - **Insertion of 3<sup>rd</sup> party IP cores reduce visibility and control.**
- **It is substantially more difficult to find a bug that is meant to be hidden.**

***Inserting malware has become much easier; because the exponential growth in design state space makes it almost impossible to find/identify malware.***

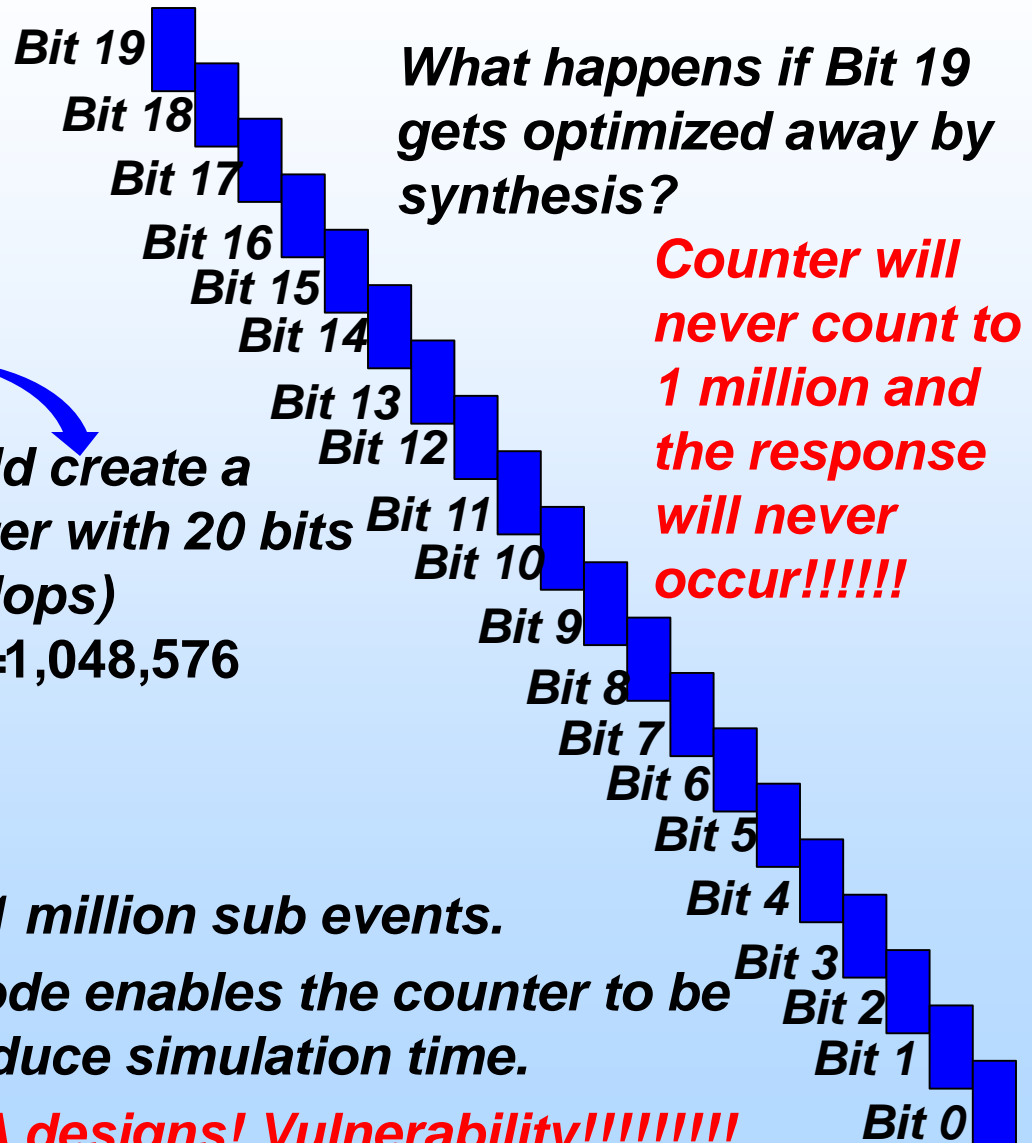
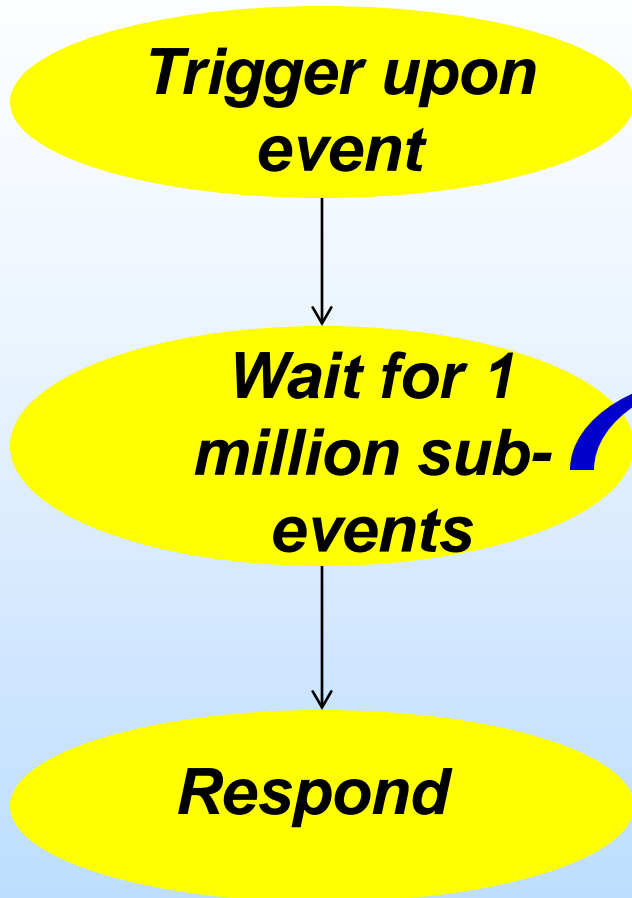
# Confronting the Verification Challenge



- The problem boils down to state space exploration ... it's just too large of a problem to tackle.
- Simulation and emulation are not sufficient... too difficult to traverse enough of a design's state space.
- In the ASIC world, rules were created:
  - Design for test (DFT).
  - Design for Verification (DFV).
  - Design for Reliability (DFR).
- In the ASIC world, DFT and DFV rules have been incorporated in library cells.
- In most cases DFR is used in FPGA design methodology.
- Most DFT and DFV rules are not used in FPGA designs because they take up too much area and compromise performance.



# Example of A Common Design Bug



What happens if Bit 19 gets optimized away by synthesis?

**Counter will never count to 1 million and the response will never occur!!!!!!**

Should create a counter with 20 bits (flip-flops)  
 $2^{20}=1,048,576$

*Might be difficult to simulate 1 million sub events.*

*ASIC Solution (DFV) : test mode enables the counter to be loaded with any number to reduce simulation time.*

**Generally not done with FPGA designs! Vulnerability!!!!!!!!!!!!**



# Common Mitigation Challenges

- **Circuits might need mitigation:**
  - Space environment or
  - Man-made radiation.
- **If required, for FPGA's that do not include embedded mitigation, the user inserts mitigation into the design.**
- **Challenges:**
  - Many users aren't aware of the proper mitigation strategy required by the target FPGA.
  - Tools are not efficient for modern FPGAs.
  - If not implemented correctly, mitigation can be optimized away.
  - IP cores are complicating the insertion process (can't insert mitigation to the inside of many of the IP cores).
  - Currently, there is no way to verify that the correct mitigation strategy was implemented .
- **Repercussion: A circuit that is expected to be mitigated, but not mitigated correctly, will not have the proper protection and will malfunction in its target environment.**

# Reducing Vulnerability: DFT, DFV, and DFR (1)



- **Enhancing the design process:**
  - Create unambiguous documentation.
  - Follow strict synchronous design rules.
  - Plan global routing control for less internal noise.
  - Carefully select and analyze I/O for less interface noise and power.
  - Create reusability rules.
  - Abide by a configuration management process.
  - Implementation of assertion based HDL.
- **ASIC specific:**
  - Include additional logic into the flip-flops to implement test mode control (scan-rings).
    - Increases control and visibility of internal logic for bug detection and verification.
    - Increases logic area because flip-flops have more logic and routing is more complex (test versus normal mode of operation).

# Reducing Vulnerability: DFT, DFV, and DFR (2)



- **Improving the simulation process:**
  - Implementation of assertion based HDL.
  - Forcing events or states.
  - Performing a mixture of random and structured tests.
  - Usage of bus functional models (BFMs).
- **Improving tool usage:**
  - Equivalence checking.
  - Area and power prediction. Area and power control (floor-planning).
  - Static timing analysis and statistical timing analysis.
  - Lint checking.
  - Formal Verification.
  - Clock domain crossing checks.
- **Design Reviews (beginning, middle, and end of design cycle).**
- **Separate the design team and verification team.**



# Conclusions

- **Following reliable design techniques helps to reduce functional bugs and trust vulnerabilities.**
- **The key is to avoid malfunction.**
- **We cannot stop designers from creating bugs and may not be able to avoid attacks:**
  - **Verification (simulation, emulation, formal methods, and reviews) are necessary to reduce the probability of malfunction.**
  - **Checking tools should be used to validate tool output.**
  - **Protection mechanisms should be put into place regarding design team participation.**
- **FPGAs are now SOCs. Design complexity has made verification a challenging task.**
  - **ASIC verification method(DFT, DFV, and formal methods) may need to be considered to assist in the verification flow.**



# Acknowledgements

- **We would like to thank the support of the NASA Electronic Parts and Packaging Program (NEPP).**
- **The authors would also like to thank Martha O'Bryan for her aid in putting this paper and its accompanying presentation together.**