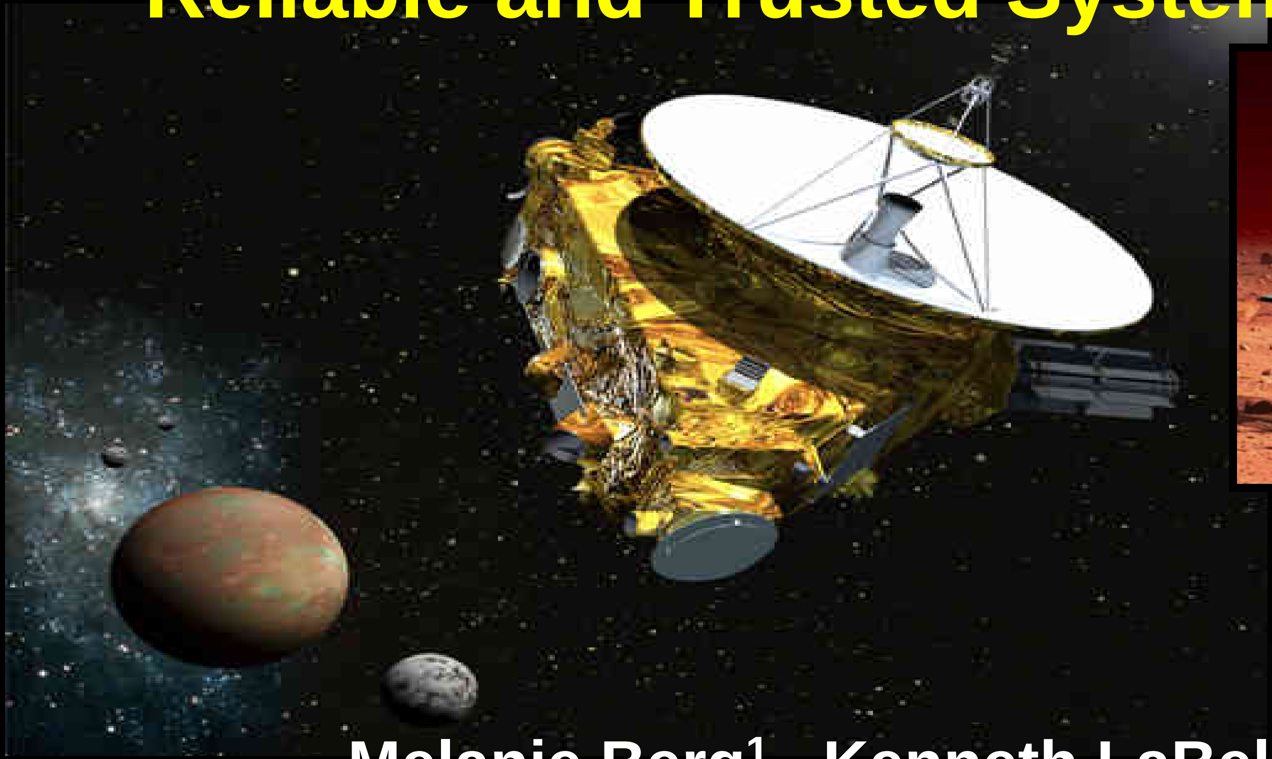
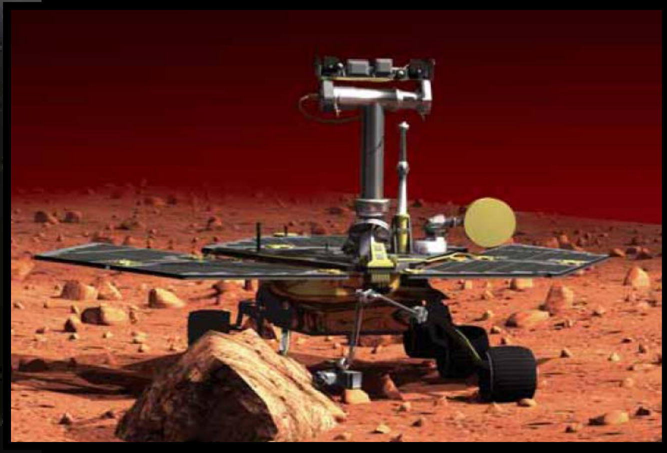
Verification of Triple Modular Redundancy (TMR) Insertion for Reliable and Trusted Systems

Melanie Berg[1], Kenneth LaBel[2]

1.AS&D in support of NASA/GSFC

Melanie.D.Berg@NASA.gov

2. NASA/GSFC

Kenneth.A.LaBel@NASA.gov

To be presented by Melanie Berg at the Microelectronics Reliability & Qualification Working Meeting (MRQW) 2016, El Segundo, CA, February 9-10, 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

- Binary decision diagram (BDD)
- Block Triple Modular Redundancy (BTMR)
- Combinatorial logic (CL)
- Computer Aided Design (CAD)
- Device under test (DUT)
- Distributed triple modular redundancy (DTMR)
- Dual interlocked storage cell (DICE)
- Edge-triggered flip-flops (DFFs)
- Equivalence checking (EC)
- Error detection and correction (EDAC)
- Error rate (dE/dt)
- Fault tolerance (FT)
- Field programmable gate array (FPGA)
- Formal verification (FV)
- Global triple modular redundancy (GTMR)
- Hardware description language (HDL)
- Input – output (I/O)
- Linear energy transfer (LET)
- Local triple modular redundancy (LTMR)
- Mitigation Window (MW)
- Mean time to failure (MTTF)
- Operational frequency (*fs)*
- Power on reset (POR)
- Radiation Effects and Analysis Group (REAG)
- Single Error Correct Double Error Detect (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)
- Static timing analysis (STA)
- Triple modular redundancy (TMR)

Abstract

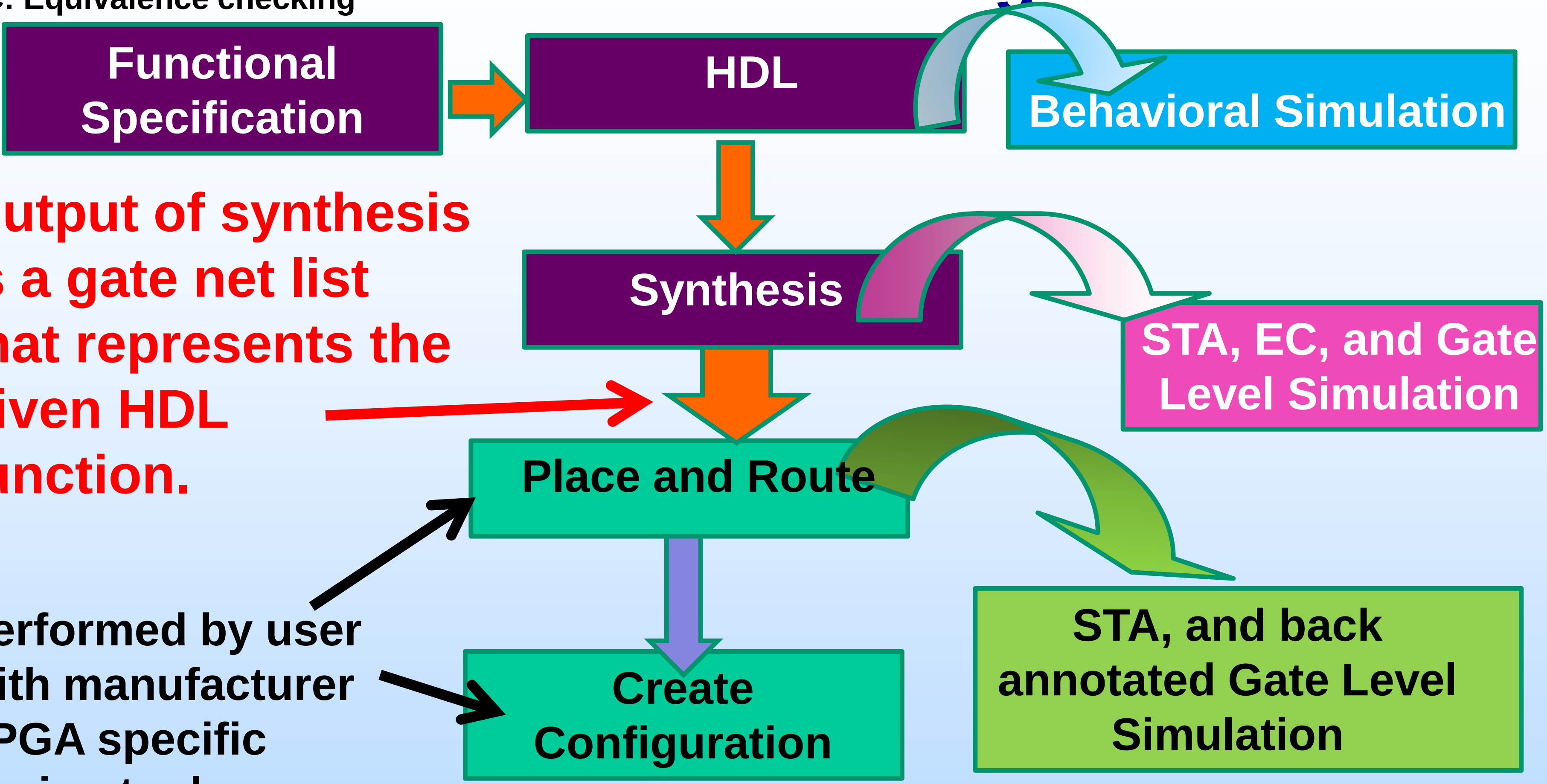
- We propose a method for triple modular redundancy (TMR) insertion verification that satisfies the process for reliable and trusted systems.
- If a system is expected to be protected using TMR, improper insertion can jeopardize the reliability and security of the system.
- Due to the complexity of the verification process and the complexity of digital designs, there are currently no available techniques that can provide complete and reliable confirmation of TMR insertion.

Overview

- This presentation addresses the challenge of confirming that TMR has been inserted without corruption of functionality and with correct application of the expected TMR topology.
- The proposed verification method combines the usage of existing formal analysis tools with a novel search-detect-and-verify tool.
- This presentation does not address the strength/performance of the selected TMR.
  - The susceptibility of a circuit post-TMR insertion should be evaluated using accelerated testing.
  - Selection of the appropriate TMR scheme is based on the device type, requirements, and design- flush frequency.

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

FPGA User Design Flow



Output of synthesis is a gate net list that represents the given HDL function.

Performed by user with manufacturer FPGA specific design tools

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

Triple Modular Redundancy (TMR)

TMR Schemes Use Majority Voting

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

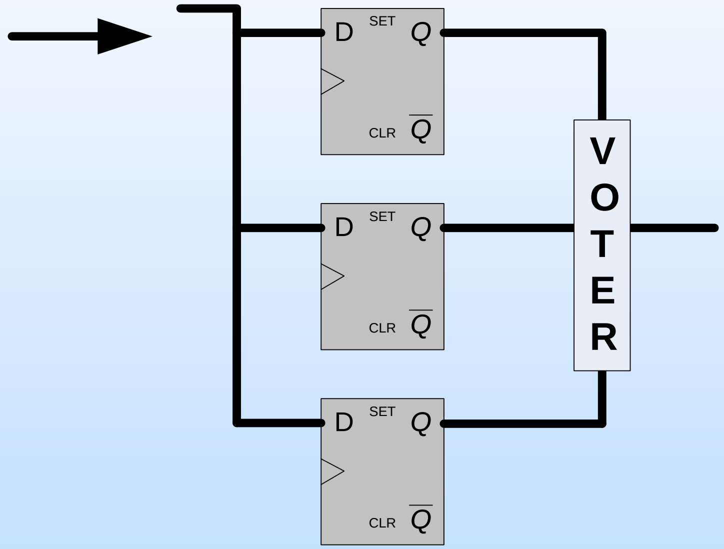
I0	I1	I2	Majority Voter
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
[illegible]	[illegible]	[illegible]	1

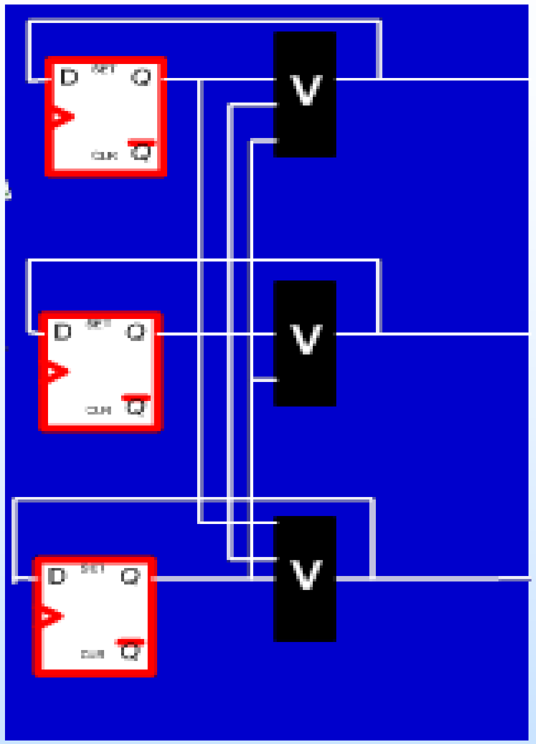
Best 2 out of 3

Triplicate and Vote

Triplicate and Vote: But Implementation Schemes Are Not All the Same

TMR Schemes are differentiated by how the triplication is performed and where the voters are placed



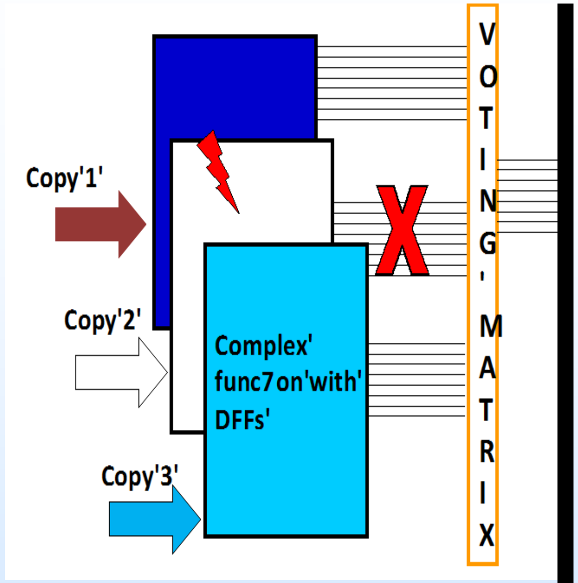


Singular Data Path: Localized Triple Modular redundancy (LTMR)

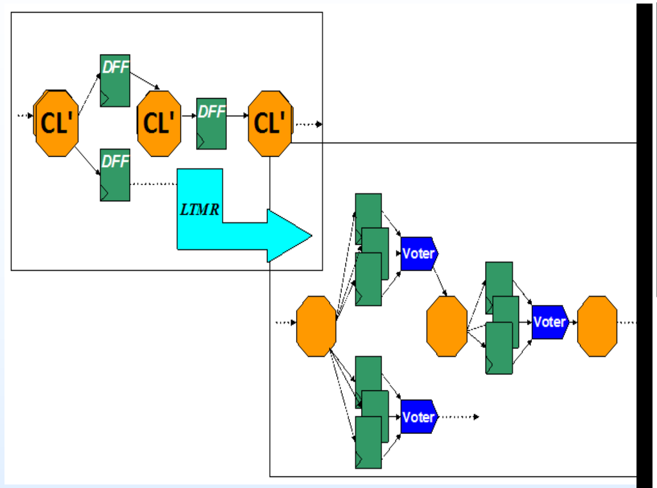
Redundant Data Path: Distributed TMR (DTMR) or Global TMR (GTMR = XTMR)

But… it's not this easy!!!!!!!!!!!!!!!!!!!!!!

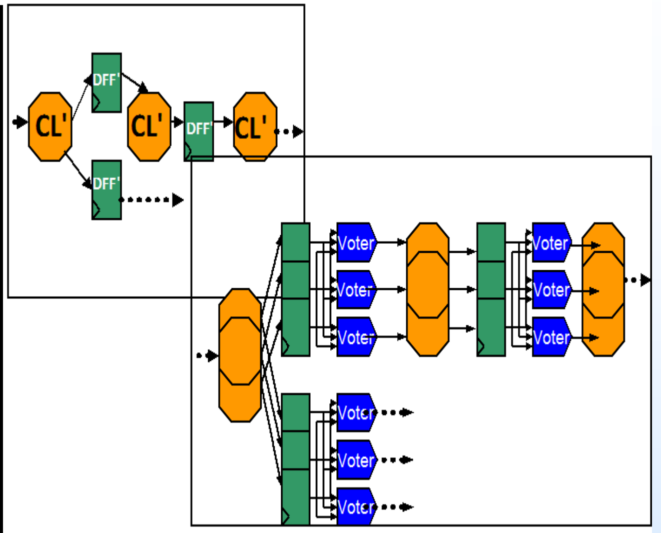
Various TMR Schemes



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.



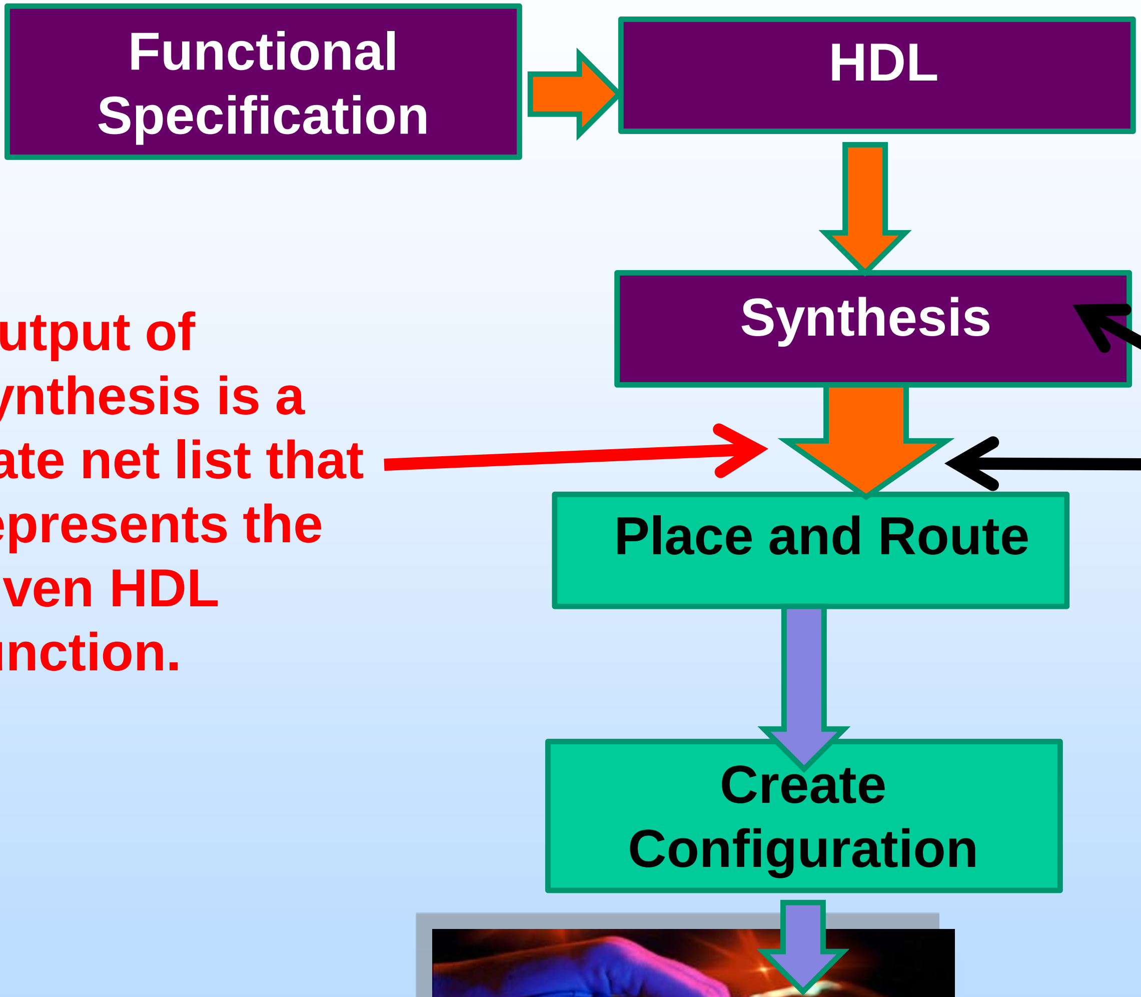
Block diagram of local TMR (LTMR): only flip-flops (DFFs) are triplicated and data-paths stay singular; voters are brought into the design and placed in front of the DFFs.



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).

HDL: Hardware description language

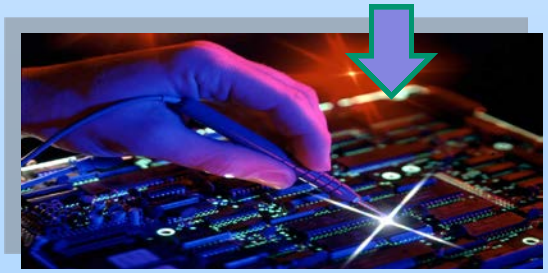
FPGA User Design Flow



Output of synthesis is a gate net list that represents the given HDL function.

TMR can be inserted during synthesis or post synthesis.

If inserted post synthesis, the gate level netlist is replicated, ripped apart, and voters + feedback are inserted.



Concerns during the TMR Insertion Process

- Synthesis TMR insertion:
  - Did the tool insert the correct type of TMR?
  - Did the user correctly select the expected type of TMR?
  - Does the tool implement the correct the topology of the expected TMR?
  - Is the functionality implemented as expected?
- Post-synthesis TMR insertion:
  - Is the functionality broken during the netlist replication and voter+feedback insertion process?
  - Is the functionality implemented as expected?
  - Only DTMR (or XTMR) is offered in available tools.
- Who is involved in the tool development process?
- How is the tool development environment controlled and verified?

Current Verification Process of TMR

- When using a tool – the designer usually assumes the insertion to be reliable.
- Not good enough:

I received a call from an organization that used synopsis to insert "TMR." They stated that the results were compatible to the design that had no mitigation.

  - This is because the version of the tool only had LTMR available.
  - LTMR should never be used with SRAM-based FPGAs. This has been reported.
  - Group didn't take into account nor verify the topology of the inserted TMR.

- Other suggestions to TMR verification:
  - Simulation,
  - Fault injection, and
  - Formal Verification.

None of these suggestions will suffice!

Verification of TMR Insertion

- Verification is two fold:
  - Verification that the functionality was not broken during the TMR insertion process.
  - Verification that the topology of the TMR scheme matches the expected methodology.
- Simulation and fault injection:
  - Although any type verification helps, simulation and fault injection is ineffective as a total proof because of its limited exploration of state-space.
  - Limited state-space traversal cannot ensure that all nodes have been mitigated as expected.
  - Reliable verification is compromised.

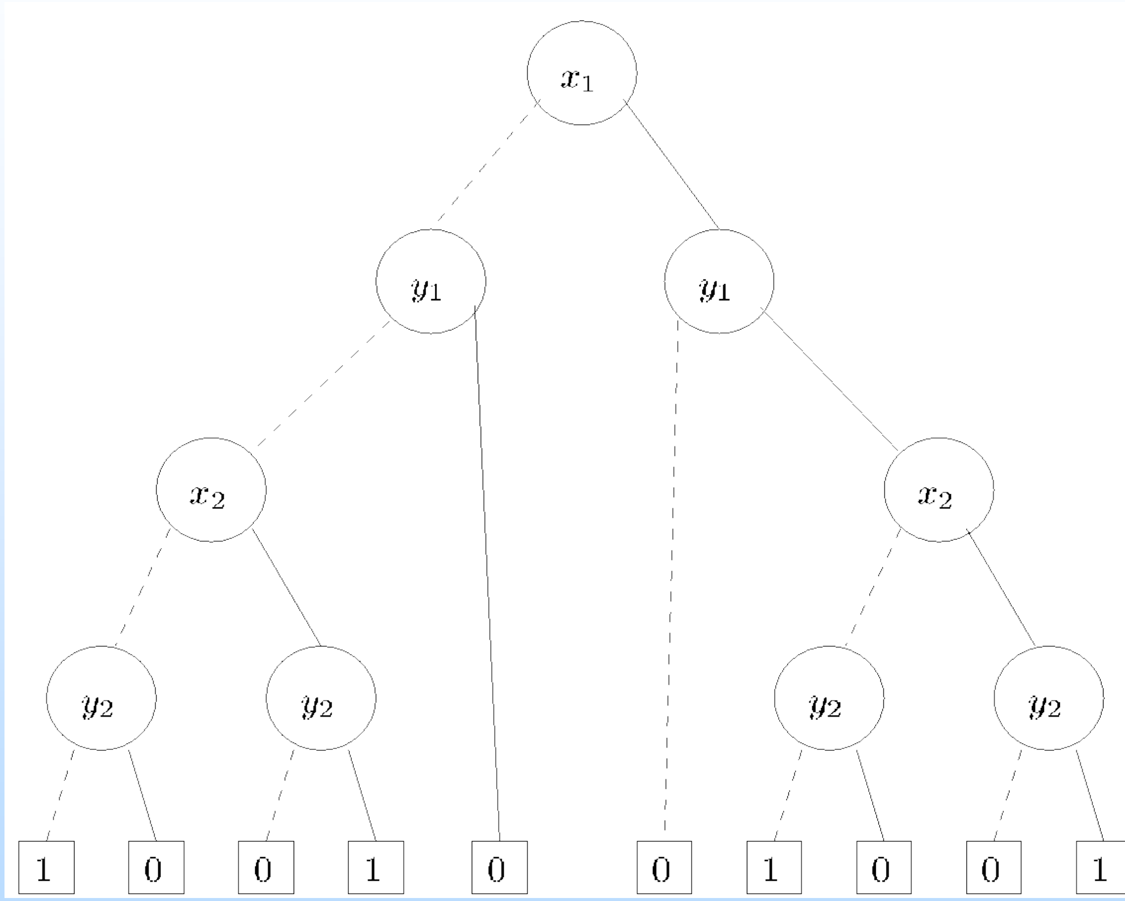
Formal Verification (FV):

Created to address the state-space traversal challenge.

Binary Decision Diagram (BDD)

Formal Verification:

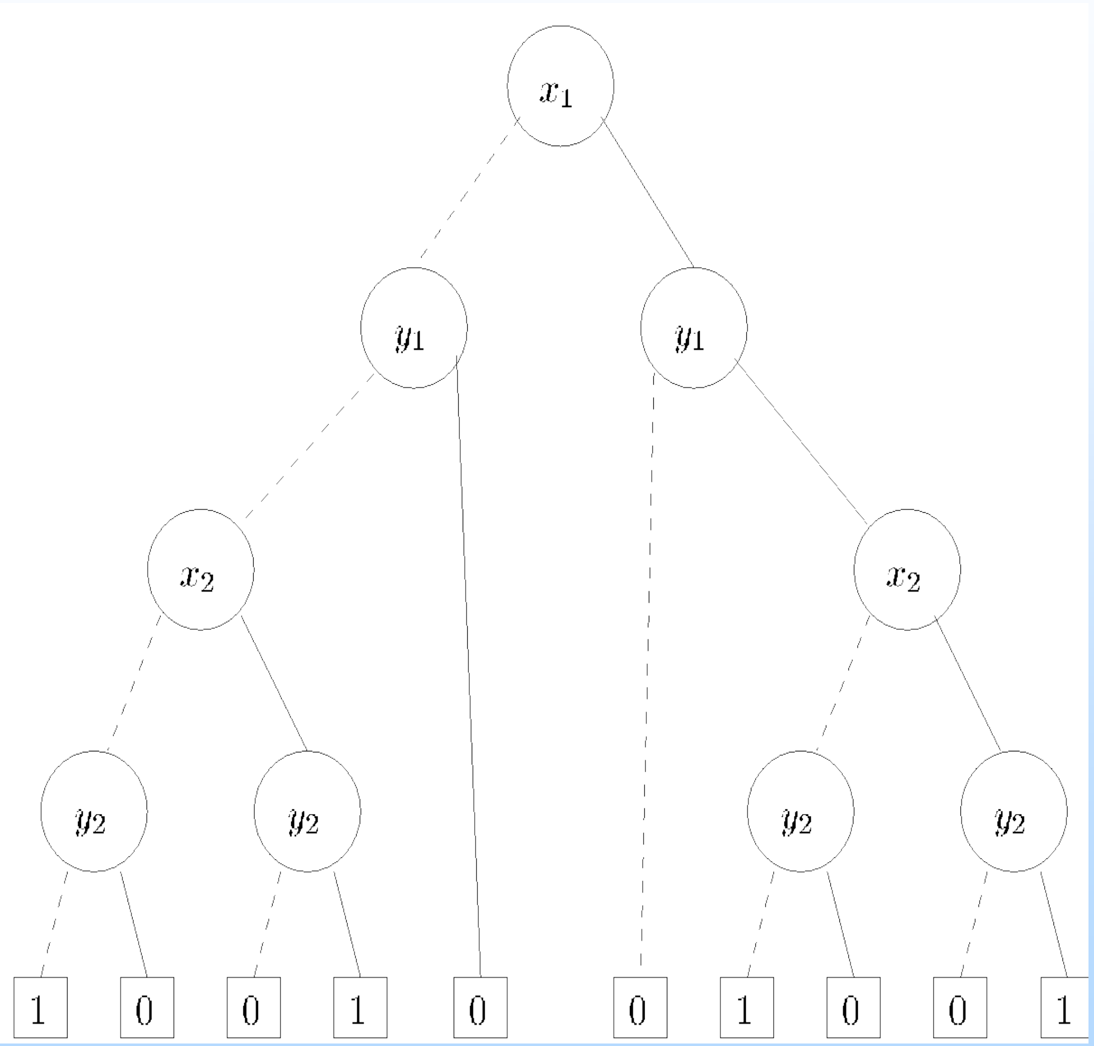
- **Synchronous circuits are modeled using BDDs.**
- **Verification of logic is performed by searching BDDs:**
  - **Find a given logic group, and**
  - **search through to terminal nodes while evaluating logic function.**
- **Types of searches are mostly event based:**
  - **A is always triggered if B occurs.**
  - **A is never triggered if B occurs.**
  - **A is triggered exactly n-cycles after B occurs.**

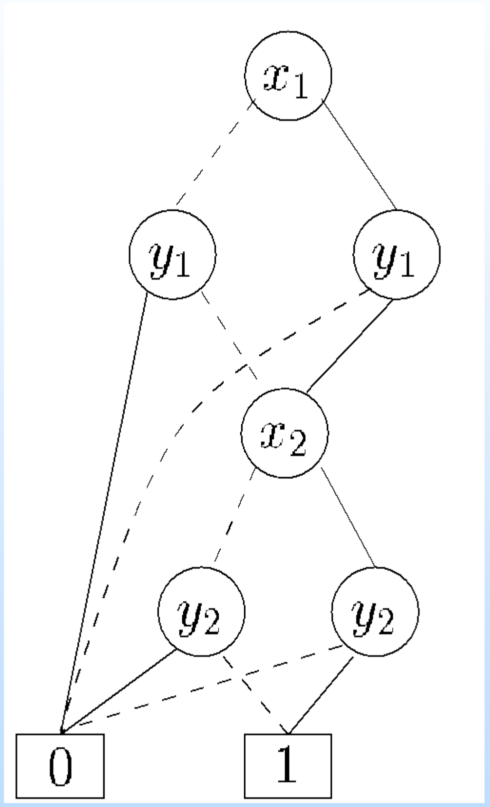


Challenges of BDD Implementation

- BDD nodes do not represent circuit elements (e.g., DFFs or combinatorial logic blocks).
- BDDs are logic representations of the circuit under evaluation.
- Logic representation of a design requires exponentially more nodes than circuit graphs.
- In most cases, full designs cannot be represented in one BDD.
- In order to model a design using BDDs:
  - Designs are intelligently partitioned, and
  - Groups of redundant logic nodes are collapsed into one group.

Collapse of Redundant Logic





Challenges of Formal Verification and TMR Verification

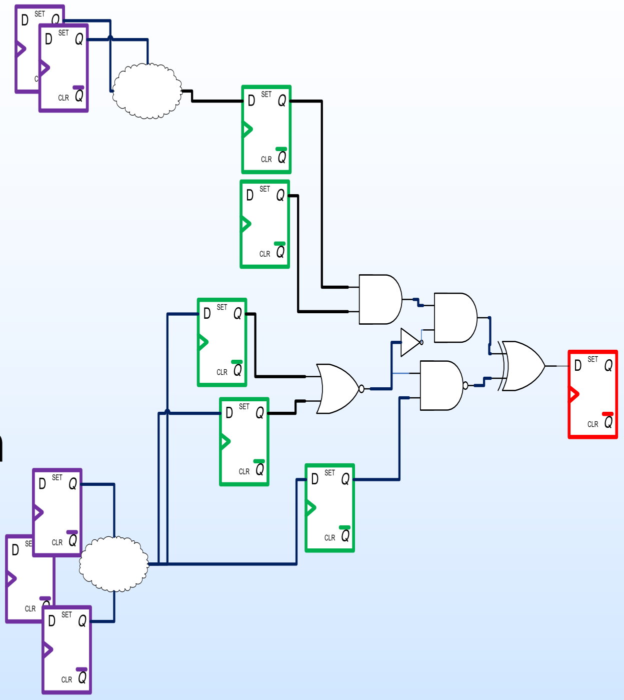
- The intention of TMR implementation is to insert redundancy.
- However, FV BDD modeling removes redundant nodes.
- Hence, FV, as currently implemented, cannot prove that:
  - redundancy exists for all nodes and
  - that it is inserted correctly.

FV and Equivalence Checking (EC)

- Equivalence checking is a “watered-down” version of FV.
- It is used to prove that the synthesized logic output matches the designer generated HDL.
- Hence, EC can be used to prove that the insertion of TMR has not broken functionality.
- However, once again, it cannot be used to verify that the expected TMR has been inserted correctly.
- EC tools are readily available.

Proposed TMR Verification Methodology

- **Functional Verification:**
  - Use EC.
- **Topology Verification:**
  - Use a circuit graph (model) to verify TMR topology with cone-of-logic theory.
  - A cone-of-logic is defined by selecting an element under analysis (vertex) and performing a backward trace towards its base components.
  - The trace starts at the cone-of-logic vertex (in this case a voter or another DFF) and ends at the vertex's previous stage of flip-flops (DFFs) or a system input (cone-of-logic base elements).

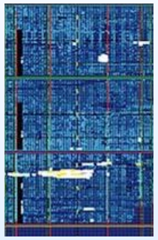


The goal is to verify whether the three copies (TMR domains) of logic that feed each voter under analysis are equivalent and the voters are inserted as expected

BTMR Topological Verification

First Stage of Algorithm: Voter and Connected DFF Evaluation

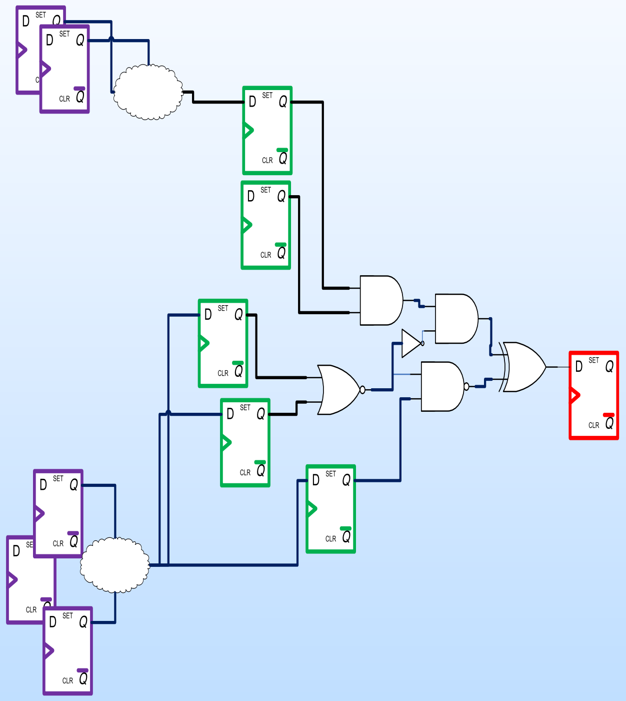
- Search for the voter under analysis; i.e., first stage vertex.
  - This voter is directly connected to a device output.
  - It will be referred to as output-voter.
- Establish the direct cone-of-logic that feeds the output-voter.
  - For proper synchronous design, it is expected that the first cone-of-logic stage consist solely of the triplicated DFFs and the output-voter.
- Verify that the cones-of-logic per TMR domain are equivalent copies of each other.
  - This step is simply checking that three of the same copies of DFFs are directly connected to the voter output.



BTMR Topological Verification

Second Stage of Algorithm: Internal Circuitry Evaluation

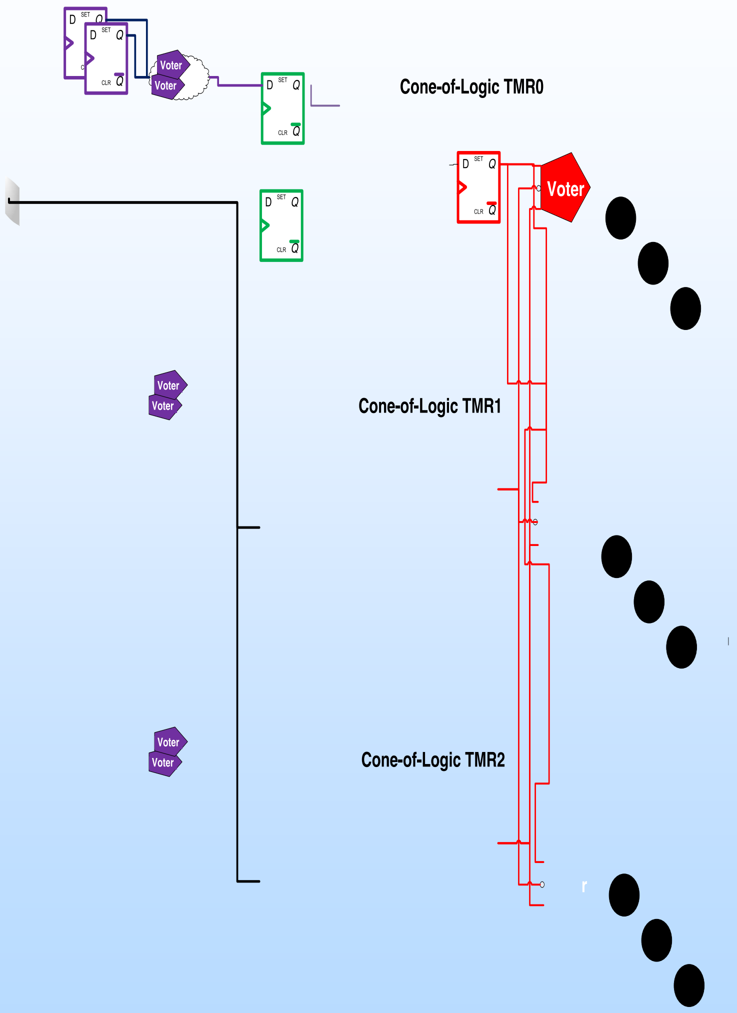
- For the following iterations, the vertex for each cone-of-logic is no longer a voter. Alternatively, the vertices are DFFs.
  - The Start Points for each cone's backward trace remain DFFs or system inputs.
  - Recursive iterations within the algorithm are based on cones-of-logic and are exhausted when all system inputs that are associated with the first stage cone-of-logic (i.e., the voter) are reached and verified.
- Perform a backwards trace to the previous stage of DFFs (cone-of-logic base components); and establish the cone-of-logic vertex (DFF) under analysis.
- Search for the two other TMR domain vertex-DFF copies and establish their cones-of-logic.
- Verify that the three cones-of-logic are equivalent.



DTMR Topological Verification

First Iteration of Algorithm:

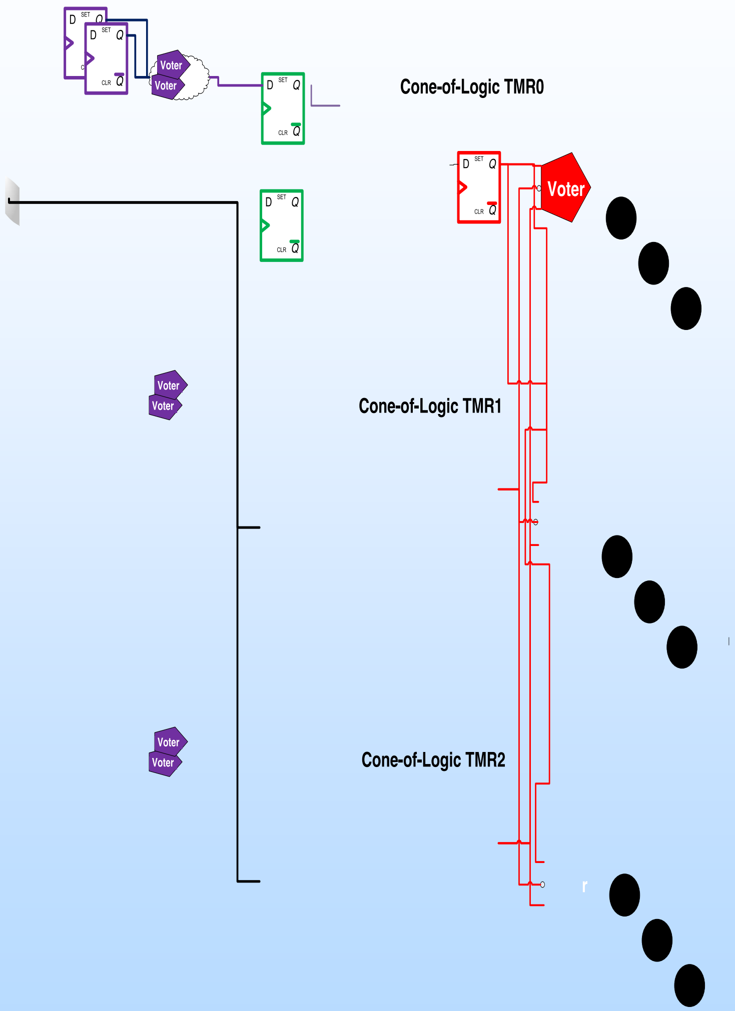
- **Identify all system output DFF-voter pairs. As a clarification, these components are physically connected to the DTMR design's output.**
- **Establish a cone-of-logic for each of the system output DFF-voter pairs.**
- **Search and identify the TMR replicas of the cones-of-logic.**
- **Compare cone-of-logic replicas across TMR domains. Verify that each cone-of-logic is equivalent to its copies.**



DTMR Topological Verification

Second Iteration of Algorithm:

- **After performing the first iteration, each of the DFF-voter pairs that were labeled as cone-of-logic base elements now become cone-of-logic vertices.**
- **The recursive procedure terminates when all paths have been traced back to system inputs.**



Conclusion

- The decision to mitigate a design with TMR is associated with a system with significantly high expectations of reliability and trust.
- Improper TMR insertion can comprise a mission along with security.
- This presentation describes a novel TMR insertion verification process with two primary concerns:
  - proof that TMR insertion leaves functionality intact; and
  - proof that TMR insertion is topologically implemented as expected.
- Computer aided design (CAD) tools exist that can verify original (unmitigated) functionality is intact. It is important to take into account that simply proving original functionality is intact does not definitively prove that mitigation is inserted correctly.
- We suggest performing an additional stage of TMR insertion verification. This stage evaluates design topology.
- We refer to the proposed topological evaluation tool as search-detect-and-verify.