Progress Seminar

An automated triple modular redundancy EDA flow for Yosys

Matt Young

11 October 2024

University of Queensland
School of Electrical Engineering and Computer Science
Supervisor: Assoc. Prof. John Williams

Table of contents

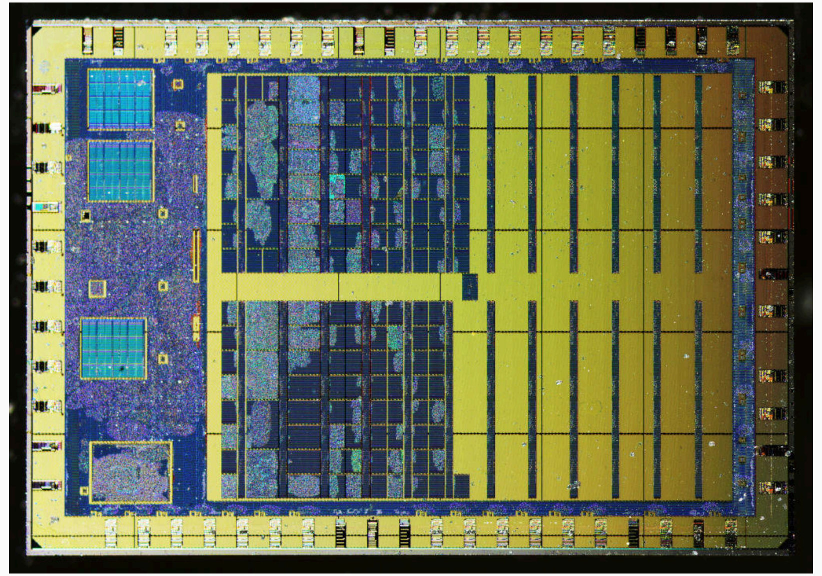
— Table of contents

Background

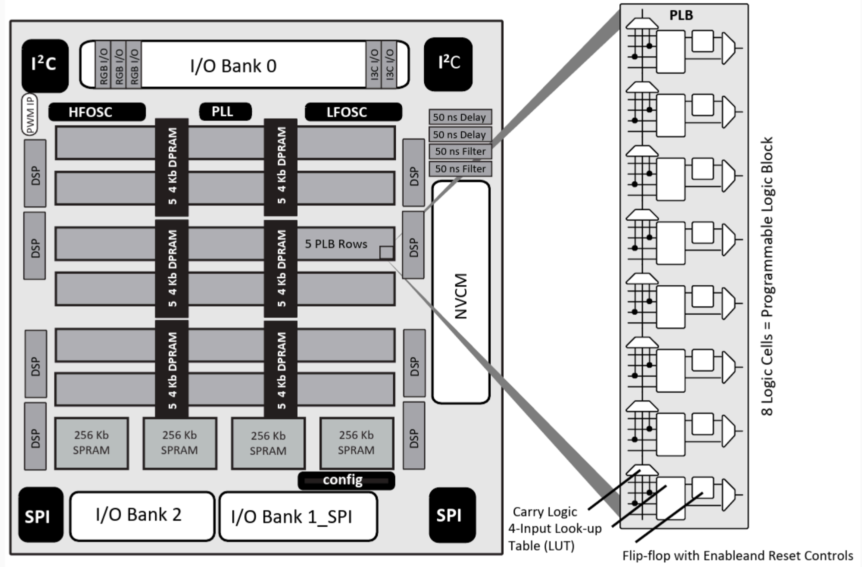
Background

Motivation

Application Specific Integrated Circuits (ASICs) and Field Programmable Gate Arrays (FPGAs) commonly deployed in space (and on Earth)...



TinyTapeout 130 nm silicon IC die shot. © 2024 Mikhail Svarichevsky, zeptobars.com (CC-BY).



Lattice iCE40 UltraPlus FPGA block diagram. © 2021 Lattice Semiconductor Corp.

Background
— Motivation

- ASICs and FPGAs deployed in space/Earth for high reliability applications

Single Event Upsets

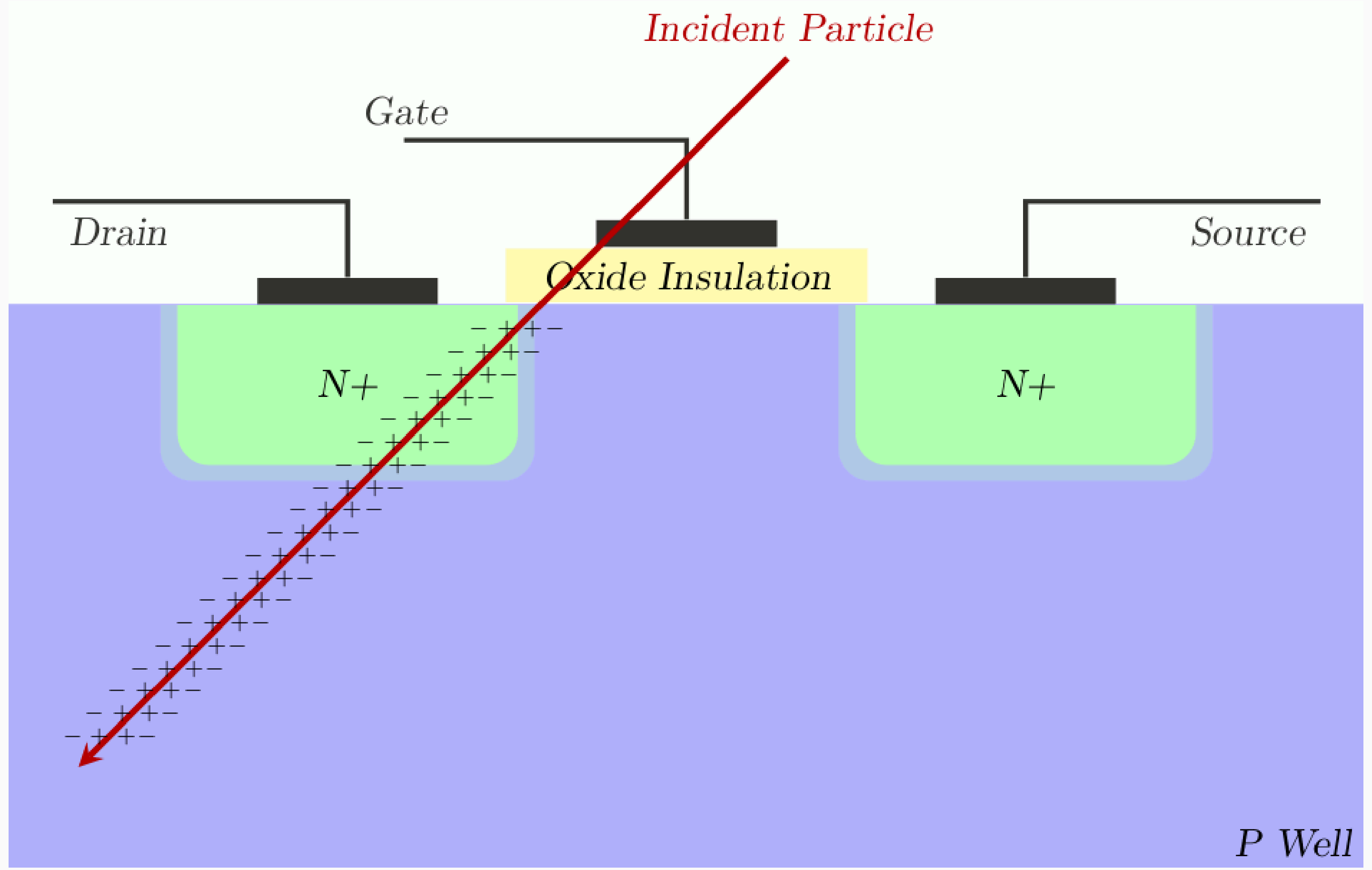


Figure 1: SEU striking a CMOS transistor. Source: https://www.cogenda.com/article/SEE

Background
— Single Event Upsets

- Suffer from SEUs caused by ionising radiation striking transistors and causing bit flips
- Particularly common for space-based applications
- Must be mitigated to prevent catastrophic failures
- Even terrestrial applications, where the Earth's magnetosphere protects chips from the majority of ionising radiation, mitigating SEUs still important for high reliability applications

SEU protection

Protection from SEUs remains expensive!

RAD750 CPU [1] (James Webb Space Telescope, Curiosity rover, + many more) is commonly used, but costs **>$200,000 USD** [2]!

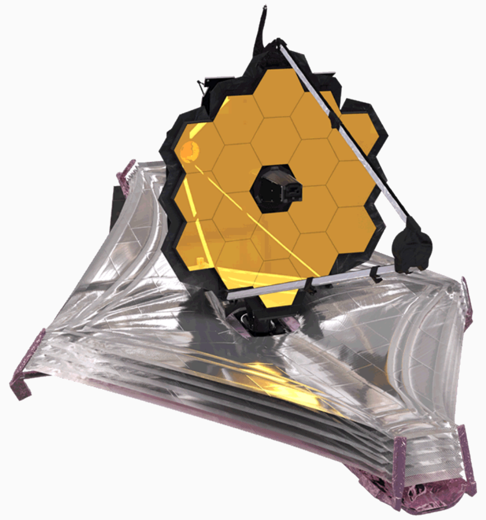


Figure 2: James Webb Space Telescope.

Source: https://commons.wikimedia.org/wiki/File:JWST_spacecraft_model_3.png

Background

— SEU protection

Triple Modular Redundancy

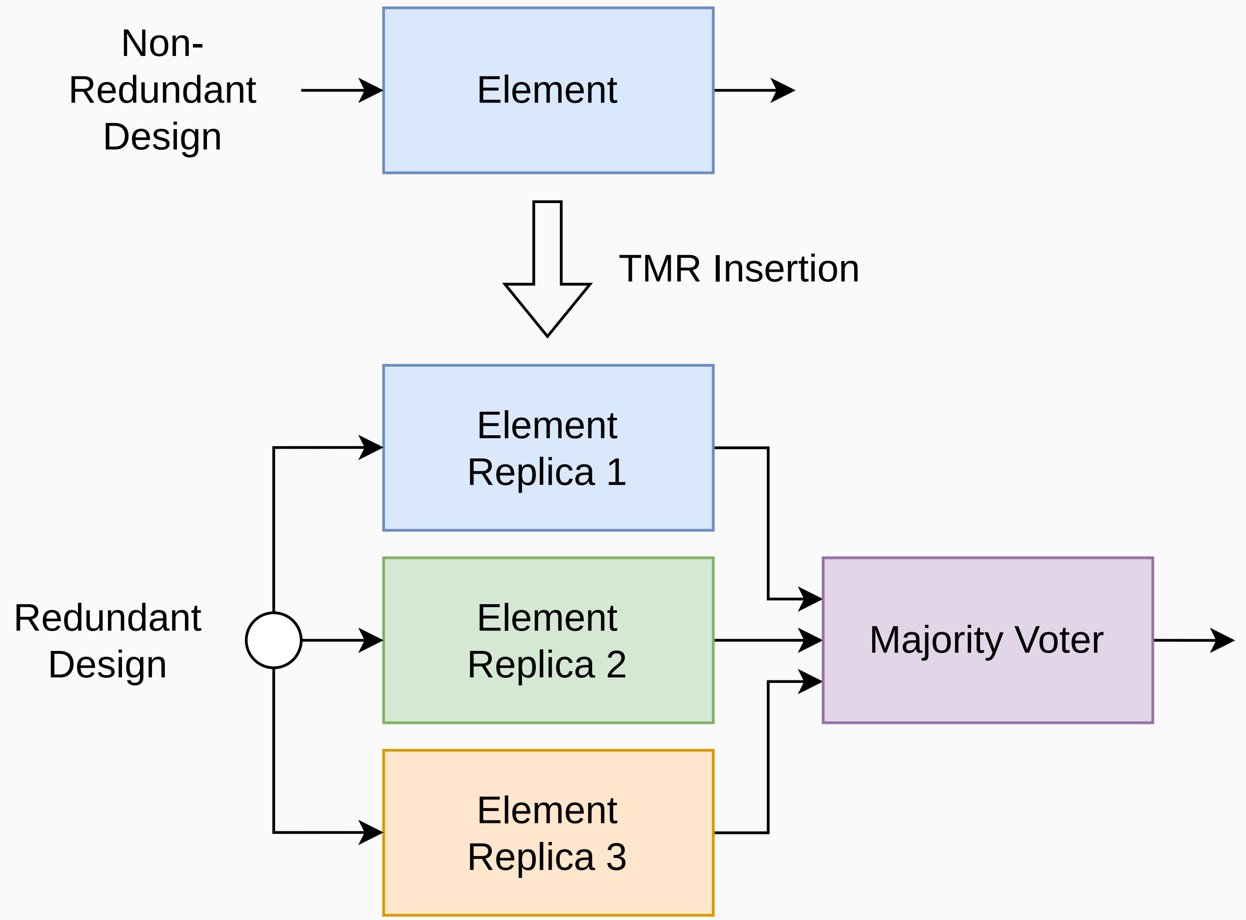


Figure 3: Diagram of TMR applied to FPGA/ASIC abstract elements (logic blocks/standard cells).

Background
— Triple Modular Redundancy

Triple Modular Redundancy

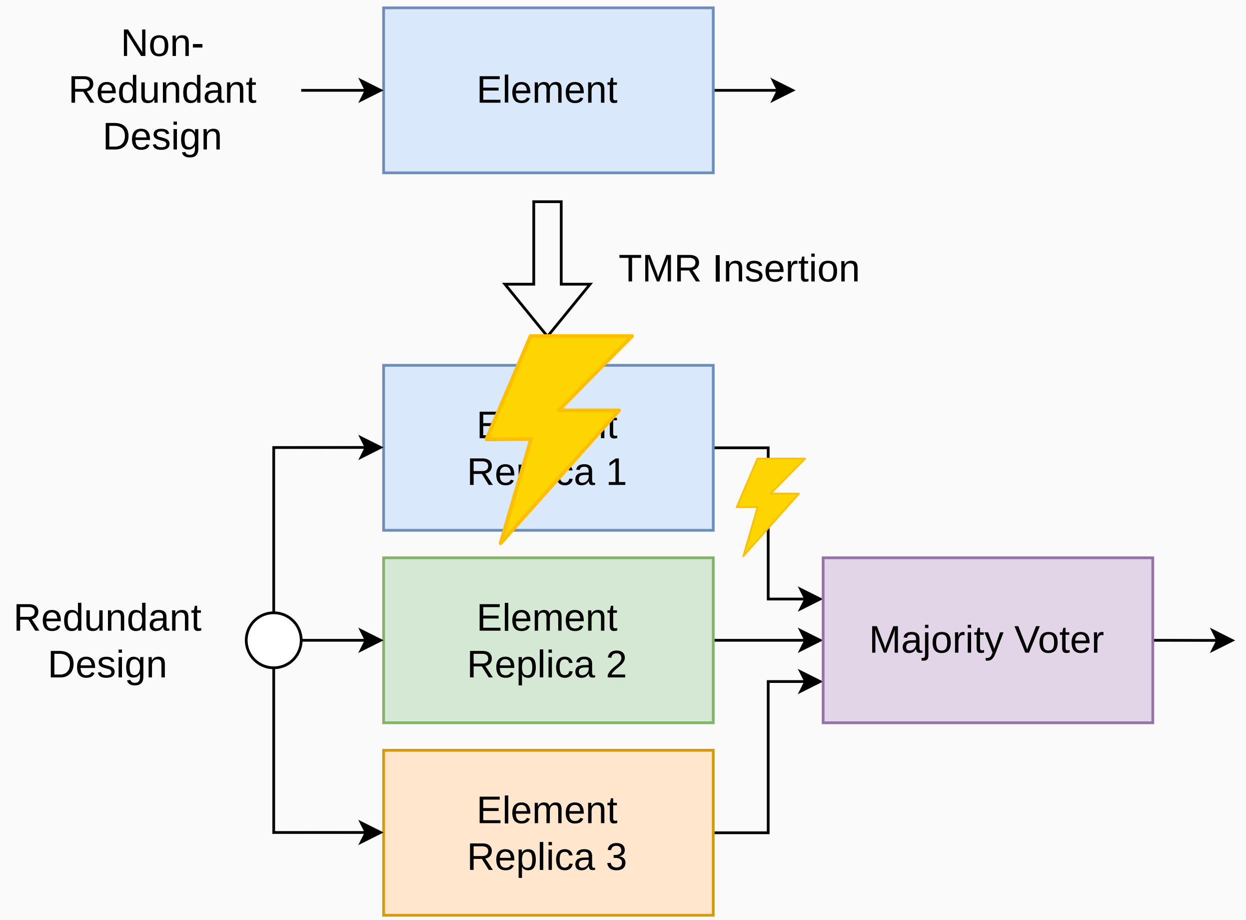


Figure 3: Diagram of TMR applied to FPGA/ASIC abstract elements (logic blocks/standard cells).

Background
— Triple Modular Redundancy

TMR can be added manually...

but this is **time consuming** and **error prone**.

Can we automate it?

Background
— Triple Modular Redundancy

TaMaRa

TaMaRa

Concept

Implement TMR as a pass in an EDA synthesis tool.

- Integrated with the rest of the flow
- Easy to use
- Fully automated

Concept

Implement TMR as a pass in an EDA synthesis tool.

- Integrated with the rest of the flow
- Easy to use
- Fully automated

Goal: Pick any design, of any complexity, “press a button” and have it be rad-hardened.

TaMaRa
— Concept

Implement TMR as a pass in an EDA synthesis tool.

- Integrated with the rest of the flow
- Easy to use
- Fully automated

Goal: Pick any design, of any complexity, “press a button” and have it be rad-hardened.

Yosys [3] is the best (and the only) open-source, research grade EDA synthesis tool.

TaMaRa
— Concept

Implement TMR as a pass in an EDA synthesis tool.

- Integrated with the rest of the flow
- Easy to use
- Fully automated

Goal: Pick any design, of any complexity, “press a button” and have it be rad-hardened.

Yosys [3] is the best (and the only) open-source, research grade EDA synthesis tool.

- Proprietary vendor tools (Synopsys, Cadence, Xilinx, etc) immediately discarded
- Can’t be extended to add custom passes

Two main paradigms:

- **Design-level approaches** (“thinking in terms of HDL”)
  - Kulis [4], Lee [5]
- **Netlist-level approaches** (“thinking in terms of circuits”)
  - Johnson [6], Benites [7], Skouson [8]

TaMaRa is mainly netlist-driven. Voter insertion is inspired by Benites [7] “logic cones” concept, and parts of Johnson [6].

Also propagate a Verilog annotation to select TMR granularity (like Kulis [4]).

Runs after techmapping (i.e. after `abc` in Yosys)

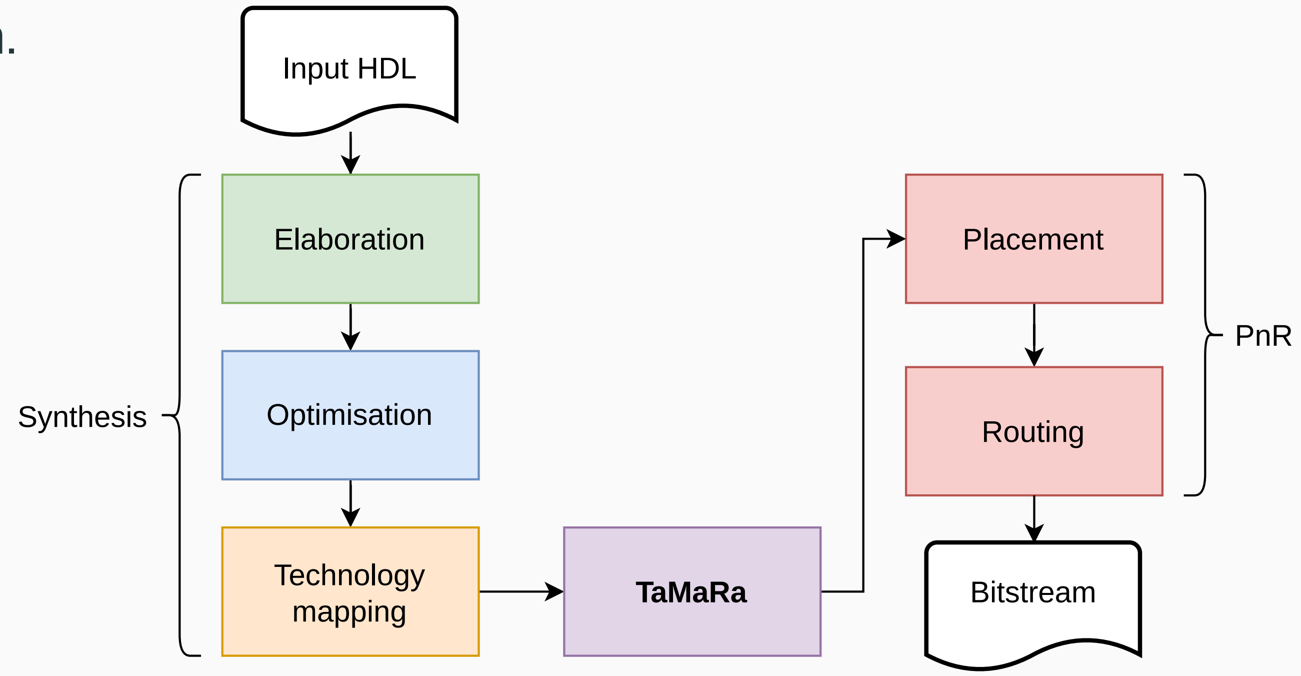


Figure 4: Location of TaMaRa plugin in Yosys EDA flow

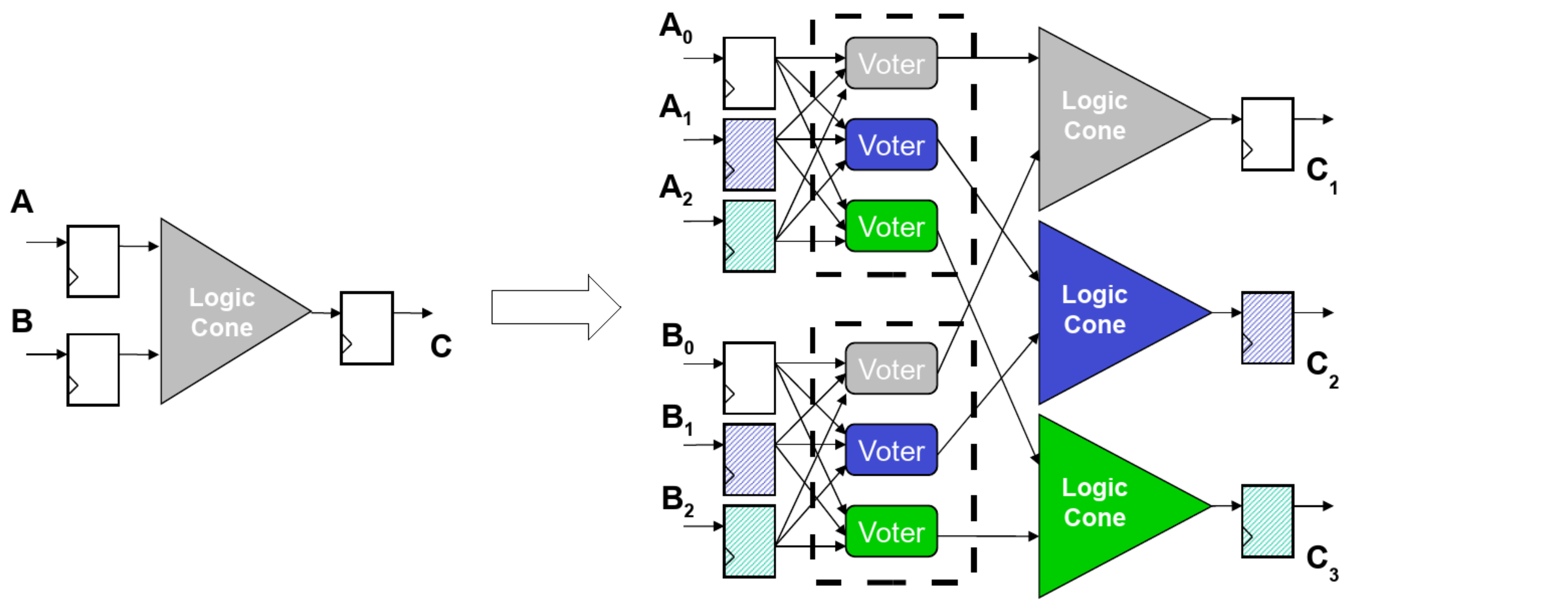


Figure 1 **A logic cone is a set of logic bounded by FFs and I/O.** When TMR is applied, each logic cone contains part of the voting logic.

Source: [9]

TaMaRa algorithm: In depth

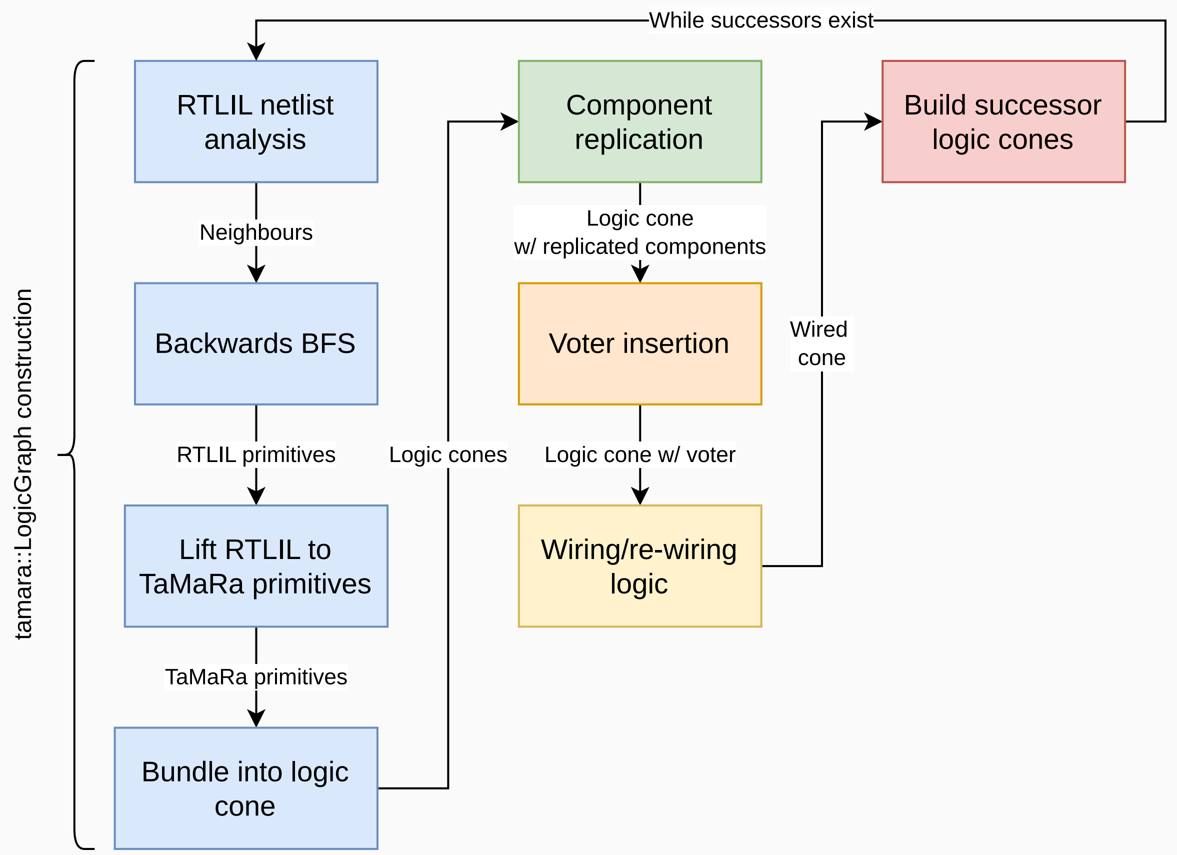


Figure 5: Description of TaMaRa plugin algorithm, as applied to Yosys RTLIL circuits.

- Construct TaMaRa logic graph and logic cones
  - Analyse Yosys RTLIL netlist
  - Perform backwards BFS from IOs to FFs (or other IOs) to collect combinatorial RTLIL primitives
  - Convert RTLIL primitives into TaMaRa primitives
  - Bundle into logic cone
- Replicate RTLIL primitives inside logic cones
- Insert voters into logic cones
- Wiring
  - Wire voter up to replicated primitives
  - Wire replicated primitive IOs to the rest of the circuit
  - Factor in feedback loop circuits
- Build successor logic cones
- Repeat until no more successors

Comprehensive verification procedure using formal methods, simulation and fuzzing.

Driven by SymbiYosys tools *eqy* and *mcy*

- In turn driven by Satisfiability Modulo Theorem (SMT) solvers (Yices [10], Boolector [11], etc)

Comprehensive verification procedure using formal methods, simulation and fuzzing.

Driven by SymbiYosys tools *eqy* and *mcy*

- In turn driven by Satisfiability Modulo Theorem (SMT) solvers (Yices [10], Boolector [11], etc)

Equivalence checking: Formally verify that the circuit is functionally equivalent before and after the TaMaRa pass.

- Ensures TaMaRa does not change the underlying behaviour of the circuit.

TaMaRa
— Verification

Verification

Comprehensive verification procedure using formal methods, simulation and fuzzing.

Driven by SymbiYosys tools *eqy* and *mcy*

- In turn driven by Satisfiability Modulo Theorem (SMT) solvers (Yices [10], Boolector [11], etc)

Equivalence checking: Formally verify that the circuit is functionally equivalent before and after the TaMaRa pass.

- Ensures TaMaRa does not change the underlying behaviour of the circuit.

Mutation: Formally verify that TaMaRa-processed circuits correct injected faults in a testbench

- Ensures TaMaRa does its job!

TaMaRa
— Verification

Fuzzing

TaMaRa must work for *all* input circuits, so we need to test at scale.

TaMaRa
— Fuzzing

TaMaRa must work for *all* input circuits, so we need to test at scale.

Idea:

1. Use Verismith [12] to generate random Verilog RTL.
2. Run TaMaRa synthesis end-to-end.
3. Use formal equivalence checking to verify the random circuits behave the same before/after TMR.

TaMaRa must work for *all* input circuits, so we need to test at scale.

Idea:

1. Use Verismith [12] to generate random Verilog RTL.
2. Run TaMaRa synthesis end-to-end.
3. Use formal equivalence checking to verify the random circuits behave the same before/after TMR.

Problem: Mutation

- Need valid testbenches for these random circuits
- Requires automatic test pattern generation (ATPG), highly non-trivial
- Future topic of further research

Simulation

I want to simulate an SEU environment.

- UQ doesn't have the capability to expose FPGAs to real radiation
- Physical verification is challenging (particularly measurement)

Simulation

I want to simulate an SEU environment.

- UQ doesn't have the capability to expose FPGAs to real radiation
- Physical verification is challenging (particularly measurement)

Use one of Verilator or Yosys' own cxxrtl to simulate a full design.

- Each simulator has different trade-offs
- Currently considering picorv32 RISC-V CPU as the Device Under Test (DUT)
- Simpler DUTs will be tested as well

Simulation

I want to simulate an SEU environment.

- UQ doesn't have the capability to expose FPGAs to real radiation
- Physical verification is challenging (particularly measurement)

Use one of Verilator or Yosys' own cxxrtl to simulate a full design.

- Each simulator has different trade-offs
- Currently considering picorv32 RISC-V CPU as the Device Under Test (DUT)
- Simpler DUTs will be tested as well

Concept:

- Iterate over the netlist, randomly consider flipping a bit every cycle
  - May be non-trivial depending on simulator
- Self-checking testbench that ensures the DUT responds correctly (e.g. RISC-V CoreMark)

TaMaRa
— Simulation

Current status & future

Current status & future

Algorithm design and planning essentially complete. Yosys internals (particularly RTLIL) understood to a satisfactory level (still learning as I go).

Current status & future
— Current status

Current status

Algorithm design and planning essentially complete. Yosys internals (particularly RTLIL) understood to a satisfactory level (still learning as I go).

C++ development well under way, approaching 1000 lines across 8 files. Using C++20.

Current status

Algorithm design and planning essentially complete. Yosys internals (particularly RTLIL) understood to a satisfactory level (still learning as I go).

C++ development well under way, approaching 1000 lines across 8 files. Using C++20.

Designed majority voters and other simple circuits in Logisim and translated to SystemVerilog HDL.

Current status

Algorithm design and planning essentially complete. Yosys internals (particularly RTLIL) understood to a satisfactory level (still learning as I go).

C++ development well under way, approaching 1000 lines across 8 files. Using C++20.

Designed majority voters and other simple circuits in Logisim and translated to SystemVerilog HDL.

Started on formal equivalence checking for TaMaRa voters and simple manually-designed combinatorial circuits.

Current status & future
— Current status

Current status

Algorithm design and planning essentially complete. Yosys internals (particularly RTLIL) understood to a satisfactory level (still learning as I go).

C++ development well under way, approaching 1000 lines across 8 files. Using C++20.

Designed majority voters and other simple circuits in Logisim and translated to SystemVerilog HDL.

Started on formal equivalence checking for TaMaRa voters and simple manually-designed combinatorial circuits.

Programming hopefully finished *around* February 2025, verification by April 2025.

Current status & future
— Current status

Original circuit:

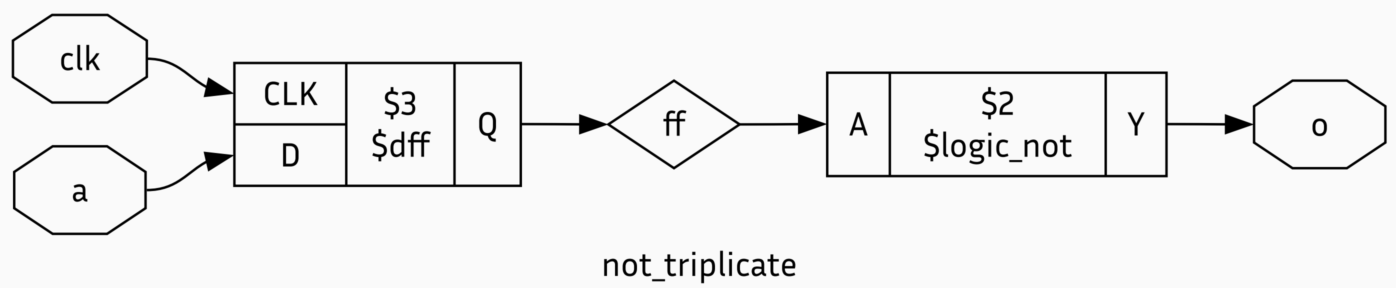


Figure 5: Yosys netlist diagram showing the result of synthesising the attached SystemVerilog RTL to logic primitives.

```
(* tamara_triplicate *)
module not_triplicate(
    input logic a,
    input logic clk,
    output logic o
);

logic ff;

always_ff @(posedge clk) begin
    ff <= a;
end

assign o = !ff;

endmodule
```

Progress: Automatically triplicating a NOT gate and inserting a voter

After `tamara_debug replicateNot`:

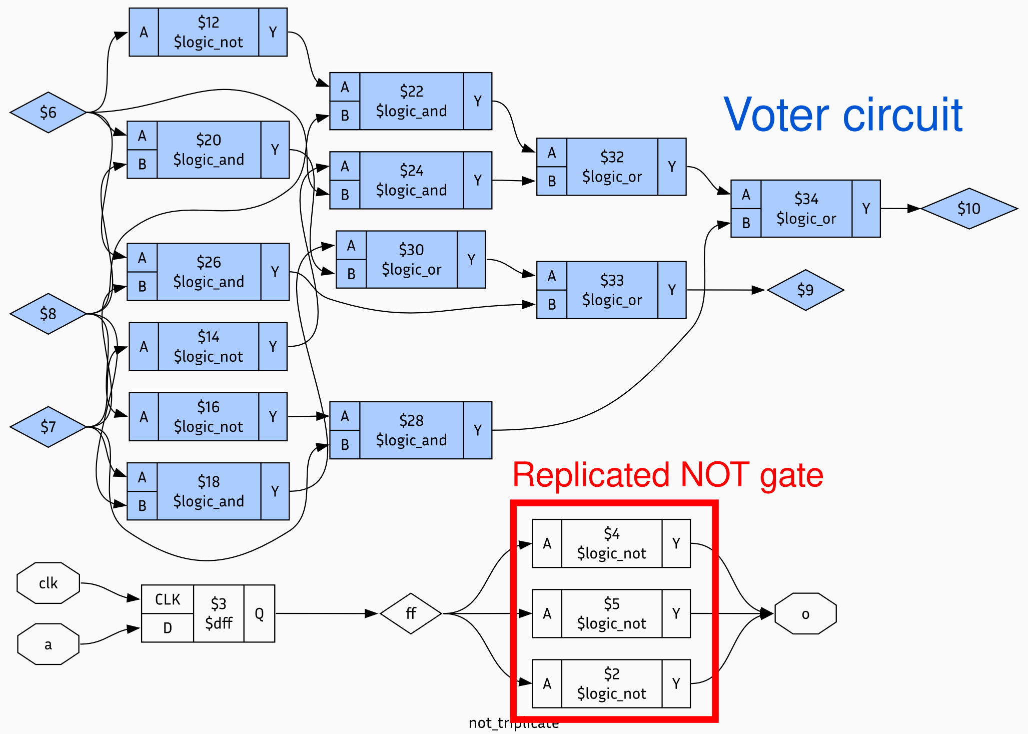


Figure 6: Netlist result of partially running TaMaRa EDA flow to identify and replicate the AND gate, and insert a voter.

Progress: Equivalence checking

Voter circuit:

a	b	c	out	err
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	1
1	0	0	0	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	0

```
module voter(
    input logic a,
    input logic b,
    input logic c,
    output logic out,
    output logic err
);
    assign out = (a && b) || (b && c) || (a && c);
    assign err = (!a && c) || (a && !b) || (b && !c);
endmodule
```

Manual design in Logisim:

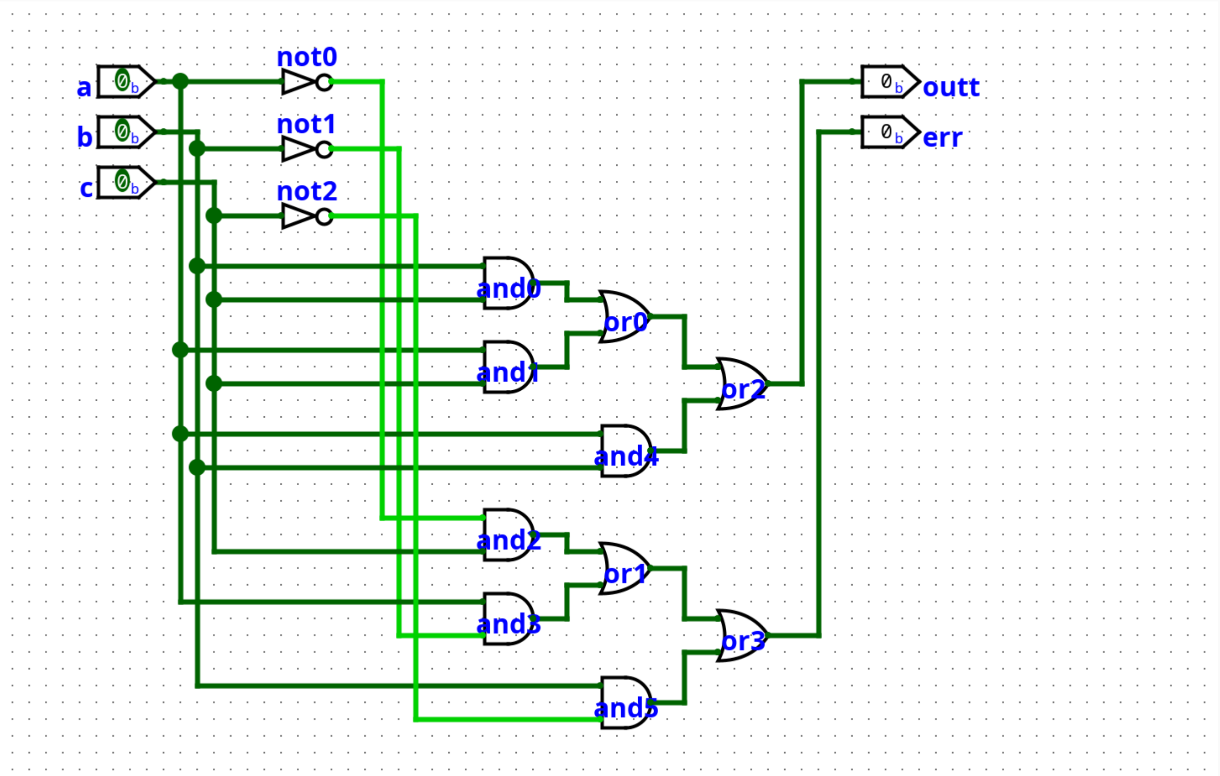


Figure 7: Manual design of previous truth table in Logisim

Current status & future
— Progress: Equivalence checking

Progress: Equivalence checking

```
    Voter
tamara::VoterBuilder::build(RTLIL::Module
*module) {
        // NOT
        // a -> not0 -> and2
        WIRE(not0, and2);
        NOT(0, a, not0_and2_wire);
        ...

        // AND
        // b, c -> and0 -> or0
        WIRE(and0, or0);
        AND(0, b, c, and0_or0_wire);
        ...

        // OR
        // and0, and1 -> or0 -> or2
        WIRE(or0, or2);
        OR(0, and0_or0_wire,
and1_or0_wire, or0_or2_wire);
        ...

        return ...;
    }
```

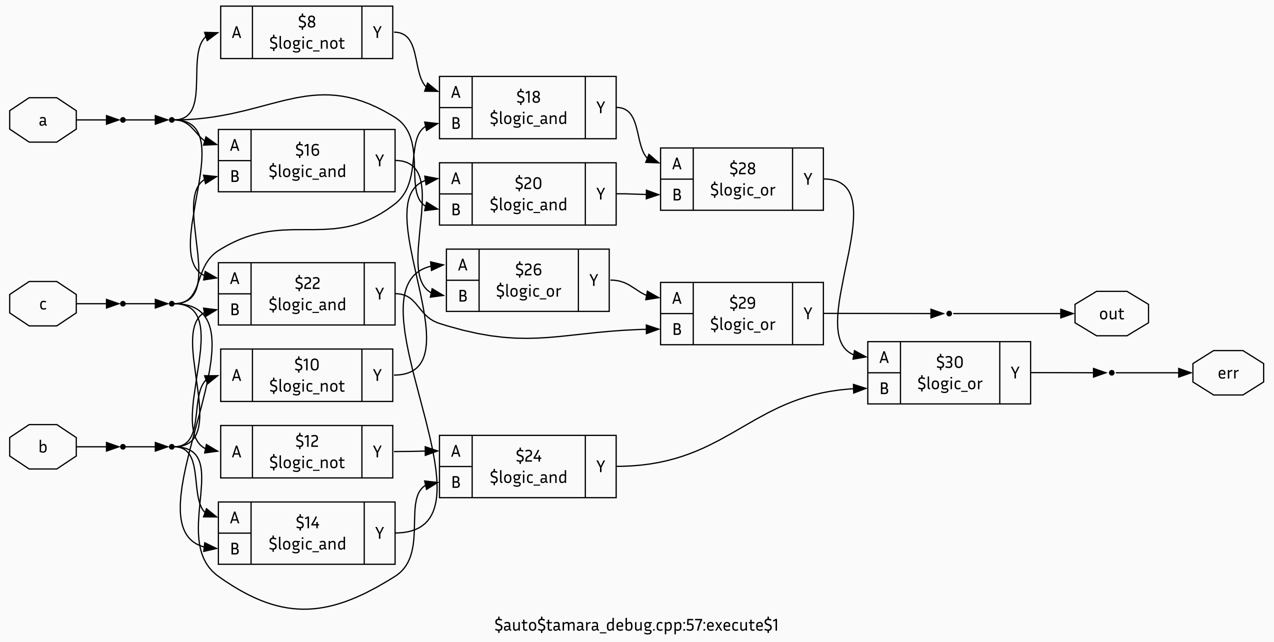


Figure 8: Yosys RTLIL netlist of voter generated by attached C++ TaMaRa plugin code.

Current status & future
— Progress: Equivalence checking

Progress: Equivalence checking

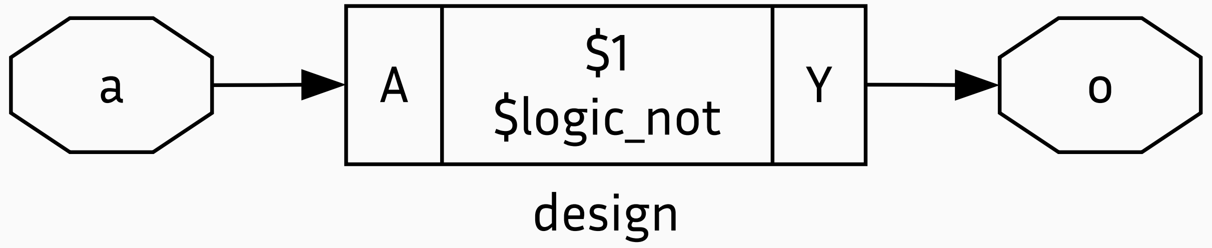
Marked equivalent by eqy in conjunction with Yices!

```
~/w/t/build (master) [n] >> eqy -f ../tests/formal/equivalence/voter.eqy
EQY 22:47:32 [voter] read_gold: starting process "yosys -ql voter/gold.log voter/gold.ys"
EQY 22:47:32 [voter] read_gold: finished (returncode=0)
EQY 22:47:32 [voter] read_gate: starting process "yosys -ql voter/gate.log voter/gate.ys"
EQY 22:47:32 [voter] read_gate: finished (returncode=0)
EQY 22:47:32 [voter] combine: starting process "yosys -ql voter/combine.log voter/combine.ys"
EQY 22:47:32 [voter] combine: finished (returncode=0)
EQY 22:47:32 [voter] partition: starting process "cd voter; yosys -ql partition.log partition.ys"
EQY 22:47:32 [voter] partition: finished (returncode=0)
EQY 22:47:32 [voter] run: starting process "make -C voter -f strategies.mk"
EQY 22:47:32 [voter] run: make: Entering directory '/home/matt/workspace/tamara/build/voter'
EQY 22:47:32 [voter] run: Running strategy 'sby' on 'voter.err'..
EQY 22:47:32 [voter] run: Proved equivalence of partition 'voter.err' using strategy 'sby'
EQY 22:47:32 [voter] run: Running strategy 'sby' on 'voter.out'..
EQY 22:47:32 [voter] run: Proved equivalence of partition 'voter.out' using strategy 'sby'
EQY 22:47:32 [voter] run: make -f strategies.mk summary
EQY 22:47:32 [voter] run: make[1]: Entering directory '/home/matt/workspace/tamara/build/voter'
EQY 22:47:32 [voter] run: make[1]: Leaving directory '/home/matt/workspace/tamara/build/voter'
EQY 22:47:32 [voter] run: make: Leaving directory '/home/matt/workspace/tamara/build/voter'
EQY 22:47:32 [voter] run: finished (returncode=0)
EQY 22:47:32 [voter] Successfully proved equivalence of partition voter.out
EQY 22:47:32 [voter] Successfully proved equivalence of partition voter.err
EQY 22:47:32 [voter] Successfully proved designs equivalent
EQY 22:47:33 [voter] summary: Elapsed clock time [H:MM:SS (secs)]: 0:00:00 (0)
EQY 22:47:33 [voter] summary: Elapsed process time [H:MM:SS (secs)]: 0:00:00 (0)
EQY 22:47:33 [voter] DONE (PASS, rc=0)
```

Figure 9: Proof of equivalence between original voter truth table and C++ plugin generated voter.

Progress: Equivalence checking (Voter insertion)

Original, very simple circuit:



Current status & future
— Progress: Equivalence checking (Voter insertion)

Progress: Equivalence checking (Voter insertion)

After manual voter insertion (using SystemVerilog):

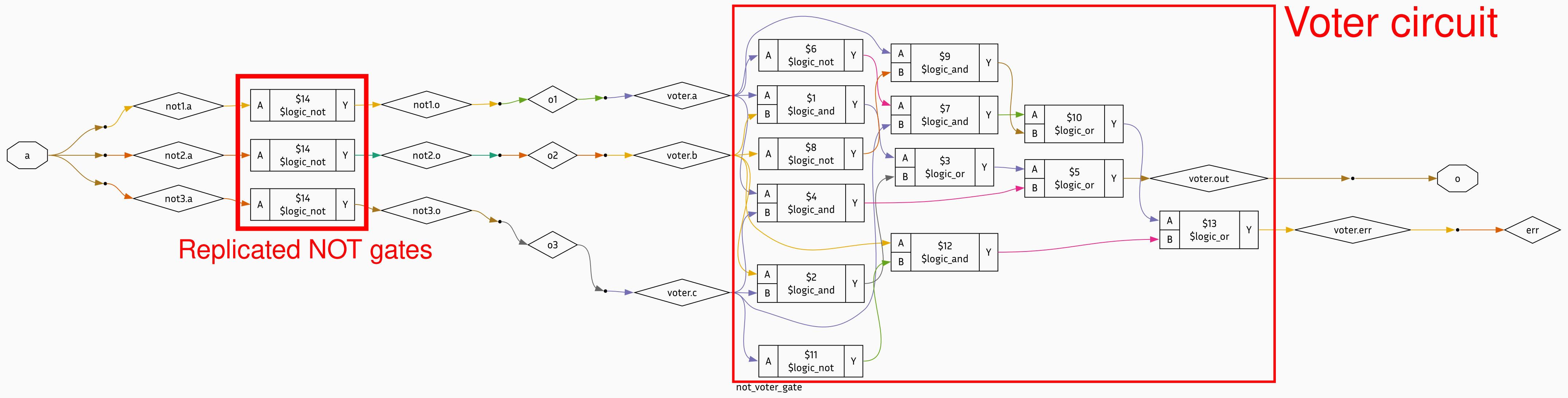


Figure 10: Netlist result of manually simulating what the TaMaRa plugin would achieve (replication + voter insertion).

Current status & future
— Progress: Equivalence checking (Voter insertion)

Progress: Equivalence checking (Voter insertion)

Are they equivalent? Yes! (Thankfully)

```
~/w/t/build (master) [n] >> eqy -f ../tests/formal/equivalence/not_voter.eqy
EQY 22:10:20 [not_voter] read_gold: starting process "yosys -ql not_voter/gold.log not_voter/gold.ys"
EQY 22:10:20 [not_voter] read_gold: finished (returncode=0)
EQY 22:10:20 [not_voter] read_gate: starting process "yosys -ql not_voter/gate.log not_voter/gate.ys"
EQY 22:10:20 [not_voter] read_gate: finished (returncode=0)
EQY 22:10:20 [not_voter] combine: starting process "yosys -ql not_voter/combine.log not_voter/combine.ys"
EQY 22:10:20 [not_voter] combine: finished (returncode=0)
EQY 22:10:20 [not_voter] partition: starting process "cd not_voter; yosys -ql partition.log partition.ys"
EQY 22:10:20 [not_voter] partition: finished (returncode=0)
EQY 22:10:20 [not_voter] run: starting process "make -C not_voter -f strategies.mk"
EQY 22:10:20 [not_voter] run: make: Entering directory '/home/matt/workspace/tamara/build/not_voter'
EQY 22:10:20 [not_voter] run: Running strategy 'sby' on 'design.o'..
EQY 22:10:20 [not_voter] run: Proved equivalence of partition 'design.o' using strategy 'sby'
EQY 22:10:20 [not_voter] run: make -f strategies.mk summary
EQY 22:10:20 [not_voter] run: make[1]: Entering directory '/home/matt/workspace/tamara/build/not_voter'
EQY 22:10:20 [not_voter] run: make[1]: Leaving directory '/home/matt/workspace/tamara/build/not_voter'
EQY 22:10:20 [not_voter] run: make: Leaving directory '/home/matt/workspace/tamara/build/not_voter'
EQY 22:10:20 [not_voter] run: finished (returncode=0)
EQY 22:10:20 [not_voter] Successfully proved equivalence of partition design.o
EQY 22:10:20 [not_voter] Successfully proved designs equivalent
EQY 22:10:20 [not_voter] summary: Elapsed clock time [H:MM:SS (secs)]: 0:00:00 (0)
EQY 22:10:20 [not_voter] summary: Elapsed process time [H:MM:SS (secs)]: 0:00:00 (0)
EQY 22:10:20 [not_voter] DONE (PASS, rc=0)
```

Figure 11: Proof of equivalence between original circuit and circuit with voter.

Progress: Equivalence checking (Voter insertion)

Are they equivalent? Yes! (Thankfully)

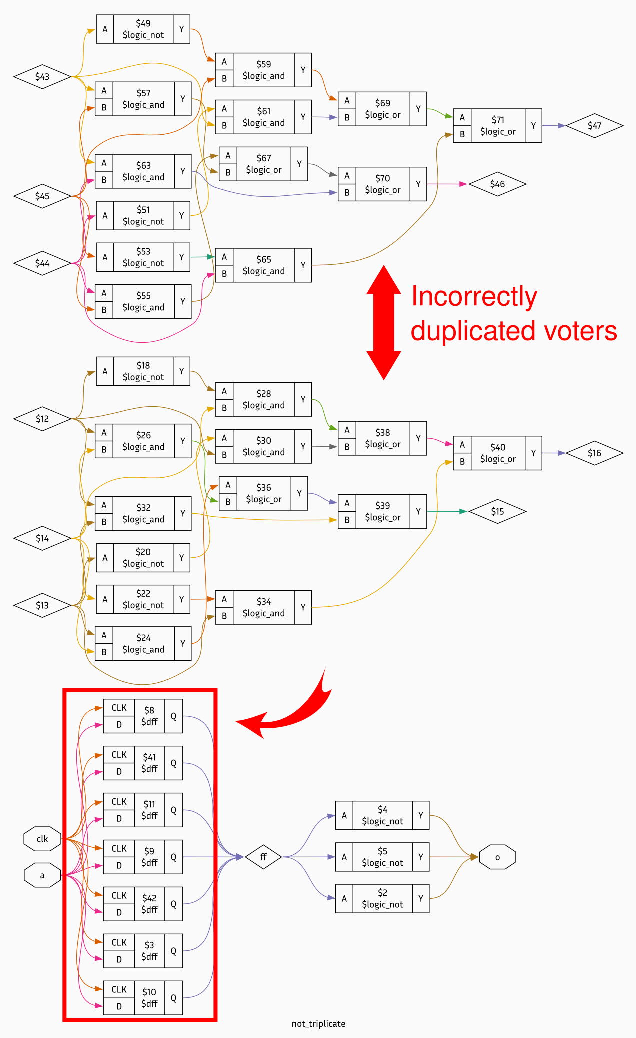
```
~/w/t/build (master) [n] ⋊> eqy -f ../tests/formal/equivalence/not_voter.eqy
EQY 22:10:20 [not_voter] read_gold: starting process "yosys -ql not_voter/gold.log not_voter/gold.ys"
EQY 22:10:20 [not_voter] read_gold: finished (returncode=0)
EQY 22:10:20 [not_voter] read_gate: starting process "yosys -ql not_voter/gate.log not_voter/gate.ys"
EQY 22:10:20 [not_voter] read_gate: finished (returncode=0)
EQY 22:10:20 [not_voter] combine: starting process "yosys -ql not_voter/combine.log not_voter/combine.ys"
EQY 22:10:20 [not_voter] combine: finished (returncode=0)
EQY 22:10:20 [not_voter] partition: starting process "cd not_voter; yosys -ql partition.log partition.ys"
EQY 22:10:20 [not_voter] partition: finished (returncode=0)
EQY 22:10:20 [not_voter] run: starting process "make -C not_voter -f strategies.mk"
EQY 22:10:20 [not_voter] run: make: Entering directory '/home/matt/workspace/tamara/build/not_voter'
EQY 22:10:20 [not_voter] run: Running strategy 'sby' on 'design.o'..
EQY 22:10:20 [not_voter] run: Proved equivalence of partition 'design.o' using strategy 'sby'
EQY 22:10:20 [not_voter] run: make -f strategies.mk summary
EQY 22:10:20 [not_voter] run: make[1]: Entering directory '/home/matt/workspace/tamara/build/not_voter'
EQY 22:10:20 [not_voter] run: make[1]: Leaving directory '/home/matt/workspace/tamara/build/not_voter'
EQY 22:10:20 [not_voter] run: make: Leaving directory '/home/matt/workspace/tamara/build/not_voter'
EQY 22:10:20 [not_voter] run: finished (returncode=0)
EQY 22:10:20 [not_voter] Successfully proved equivalence of partition design.o
EQY 22:10:20 [not_voter] Successfully proved designs equivalent
EQY 22:10:20 [not_voter] summary: Elapsed clock time [H:MM:SS (secs)]: 0:00:00 (0)
EQY 22:10:20 [not_voter] summary: Elapsed process time [H:MM:SS (secs)]: 0:00:00 (0)
EQY 22:10:20 [not_voter] DONE (PASS, rc=0)
```

Figure 12: Proof of equivalence between original circuit and circuit with voter.

Caveat: Still need to verify circuits with more complex logic (i.e. DFFs).

Current status & future
— Progress: Equivalence checking (Voter insertion)

Current problem: Duplicate DFFs




```
7.2. Computing logic graph
Module has 1 output ports, 2 selected cells

Searching from output port o
Starting search for cone 0
    ... [snip] ...
Search complete for cone 0, have 3 items

Replicating 3 collected items for logic cone 0
    Replicating ElementCellNode $logic_not$../tests/verilog/
not_triplicate.sv:16$2
    Replicating ElementWireNode ff
    Replicating FFNode $procdff$3
Checking terminals
Input node $procdff$3 is not IONode, replicating it
    Replicating FFNode $procdff$3
Warning: When replicating FFNode $procdff$3 in cone 0: Already
replicated in logic cone 0
Input node o is IONode, it will NOT be replicated

Inserting voter into logic cone 0

... [snip] ...
```


Current status & future
— Current problem: Duplicate DFFs

The future

Tasks that remain (more or less):

- Fixing duplicate logic elements when replicating RTLIL primitives
- Wiring voter to logic elements, and wiring replicated logic elements to the rest of the circuit
- Considering wiring for feedback circuits *(expected to be complex/massive time sink!)*
- Global routing of error signal to a net
- Processing complex circuits like picorv32
- Writing a cycle-accurate fault-injection simulator, and associated testbenches
- Formal equivalence checking for complex circuits
- Formal mutation coverage
- Fuzzing *(if time permits)*

Current status & future
— The future

The future

I'm aiming to produce at least one academic publication from this thesis.

- If TaMaRa works, its hybrid algorithm addresses a number of limitations in previous literature
- May be useful for research labs (CubeSats) and industry

Current status & future
— The future

The future

I'm aiming to produce at least one academic publication from this thesis.

- If TaMaRa works, its hybrid algorithm addresses a number of limitations in previous literature
- May be useful for research labs (CubeSats) and industry

TaMaRa plugin code and tests will be released open-source under the MPL 2.0 (used by Firefox, Eigen, etc). Papers will hopefully be available under CC-BY.

TaMaRa will be freely available for anyone to use and build on. Combination of academic publication + open source for widest possible reach.

Current status & future
— The future

The future

I'm aiming to produce at least one academic publication from this thesis.

- If TaMaRa works, its hybrid algorithm addresses a number of limitations in previous literature
- May be useful for research labs (CubeSats) and industry

TaMaRa plugin code and tests will be released open-source under the MPL 2.0 (used by Firefox, Eigen, etc). Papers will hopefully be available under CC-BY.

TaMaRa will be freely available for anyone to use and build on. Combination of academic publication + open source for widest possible reach.

I have also spoken with the team at YosysHQ GmbH and Sandia National Laboratories, who are very interested in the results of this project and its applications.

Conclusion

Conclusion

- TaMaRa: Automated triple modular redundancy EDA flow for Yosys
- Fully integrated into Yosys suite
- Takes any circuit, helps to prevent it from experiencing SEUs by adding TMR
- Synthesises netlist-driven approaches [9], [6] with design-level approaches [4]
- **Key goal:** "Click a button" and have any circuit run in space/in high reliability environments!

I'd like to extend my gratitude to N. Engelhardt of YosysHQ, the team at Sandia National Laboratories, and my supervisor Assoc. Prof. John Williams for their support and interest during this thesis so far.

Conclusion
— Summary

References

[1] R. Berger *et al.*, "The RAD750™ - a radiation hardened PowerPC™ processor for high performance spaceborne applications," in *2001 IEEE Aerospace Conference Proceedings (Cat. No.01TH8542)*, 2001, pp. 2263–2272. doi: 10.1109/AERO.2001.931184.

[2] H. Hagedoorn, "NASA Perseverance rover 200 MHZ CPU costs $200K." Accessed: Aug. 20, 2024. [Online]. Available: https://www.guru3d.com/story/nasa-perseverance-rover-200-mhz-cpu-costs-200k/

[3] C. Wolf and J. Glaser, "Yosys - A Free Verilog Synthesis Suite," in *Proceedings of Austrochip 2013*, 2013. [Online]. Available: http://yosyshq.net/yosys/files/yosys-austrochip 2013.pdf

[4] S. Kulis, "Single Event Effects mitigation with TMRG tool," *Journal of Instrumentation*, vol. 12, no. 1, p. C01082–C01082, Jan. 2017, doi: 10.1088/1748-0221/12/01/c01082.

[5] G. Lee, D. Agiakatsikas, T. Wu, E. Cetin, and O. Diessel, "TLegUp: A TMR Code Generation Tool for SRAM-Based FPGA Applications Using HLS," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2017, pp. 129–132. doi: 10.1109/FCCM.2017.57.

[6] J. M. Johnson and M. J. Wirthlin, "Voter insertion algorithms for FPGA designs using triple modular redundancy," in *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*, in FPGA '10. ACM, Feb. 2010. doi: 10.1145/1723112.1723154.

[7] L. A. C. Benites and F. L. Kastensmidt, "Automated design flow for applying Triple Modular Redundancy (TMR) in complex digital circuits," in *2018 IEEE 19th Latin-American Test Symposium (LATS)*, 2018, pp. 1–4. doi: 10.1109/LATW.2018.8349668.

[8] D. Skouson, A. Keller, and M. Wirthlin, "Netlist Analysis and Transformations Using SpyDrNet," in *Proceedings of the 19th Python in Science Conference*, M. Agarwal, C. Calloway, D. Niederhut, and D. Shupe, Eds., 2020, pp. 40–47. doi: 10.25080/Majora-342d178e-006.

[9] G. Beltrame, "Triple Modular Redundancy verification via heuristic netlist analysis," *PeerJ Computer Science*, vol. 1, p. e21, Aug. 2015, doi: 10.7717/peerj-cs.21.

[10] B. Dutertre, "Yices 2.2," in *International Conference on Computer Aided Verification*, 2014, pp. 737–744.

[11] A. Niemetz, M. Preiner, and A. Biere, "Boolector 2.0," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 9, no. 1, pp. 53–58, 2014.

[12] Y. Herklotz and J. Wickerson, "Finding and Understanding Bugs in FPGA Synthesis Tools," in *ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays*, in FPGA '20. Seaside, CA, USA: ACM, 2020. doi: 10.1145/3373087.3375310.

Conclusion
— References

Thank you! Any questions?