

# TaMaRa:

## An automated triple modular redundancy EDA flow for Yosys

---

Matt Young

28 August 2024

University of Queensland

Prepared for YosysHQ and Sandia National Laboratories

# Table of contents

1. Background
2. TaMaRa methodology
3. Current status & future
4. Conclusion

# Background



# About me



Hi! I'm Matt Young, 21 years old from Brisbane, Australia.

Graduated Bachelor of Computer Science earlier in 2024 from the University of Queensland.

Currently studying Bachelor of Computer Science (Honours) at UQ, involves one year research thesis.

Despite the email, thesis is **not** associated with my employer (Emesent).

# Motivation

Fault tolerant computing is important for safety critical sectors (aerospace, defence, medicine, etc.)

# Motivation

Fault tolerant computing is important for safety critical sectors (aerospace, defence, medicine, etc.)

For space-based applications, Single Event Upsets (SEUs) are very common

- Must be mitigated to prevent catastrophic failure
- Caused by ionising radiation

Even in terrestrial applications, SEUs can still occur

- Must be mitigated for high reliability applications

# Motivation

Fault tolerant computing is important for safety critical sectors (aerospace, defence, medicine, etc.)

For space-based applications, Single Event Upsets (SEUs) are very common

- Must be mitigated to prevent catastrophic failure
- Caused by ionising radiation

Even in terrestrial applications, SEUs can still occur

- Must be mitigated for high reliability applications

ASICs and FPGAs commonly deployed in space (and on Earth)...

# Motivation

Fault tolerant computing is important for safety critical sectors (aerospace, defence, medicine, etc.)

For space-based applications, Single Event Upsets (SEUs) are very common

- Must be mitigated to prevent catastrophic failure
- Caused by ionising radiation

Even in terrestrial applications, SEUs can still occur

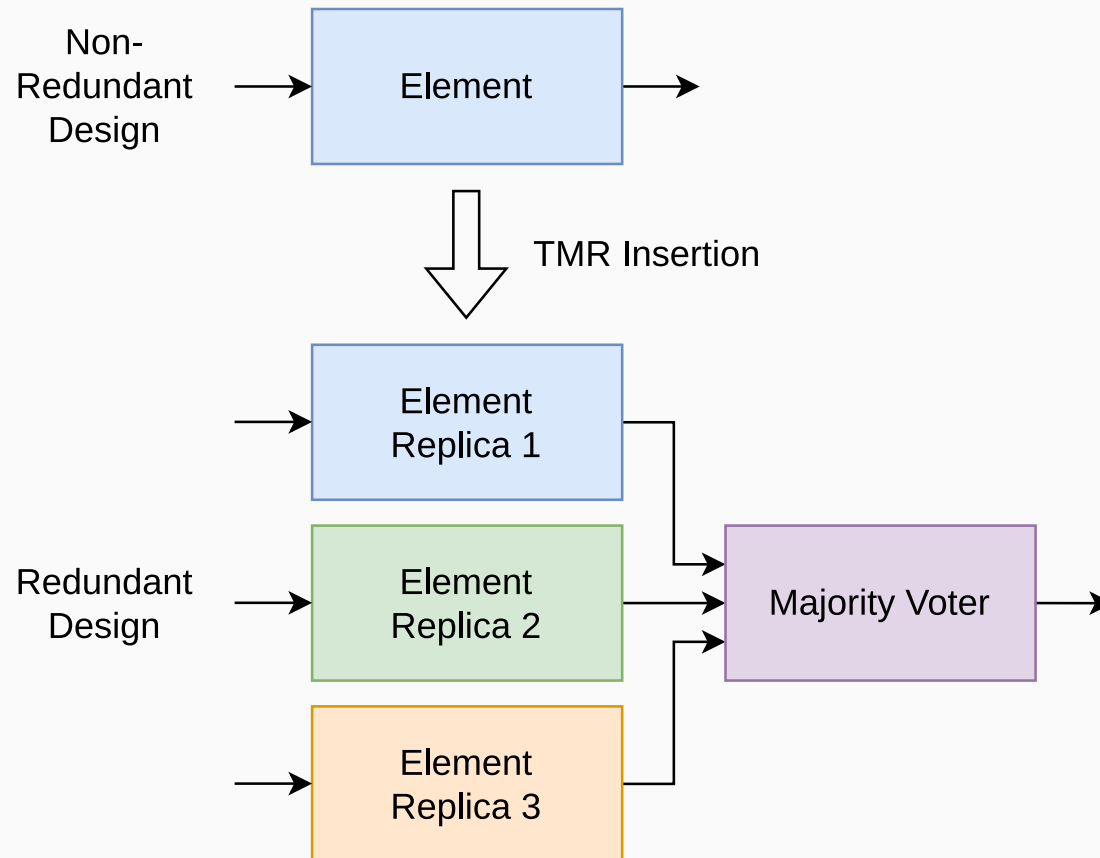
- Must be mitigated for high reliability applications

ASICs and FPGAs commonly deployed in space (and on Earth)... but protection from SEUs remains expensive!

RAD750 CPU [\[1\]](#) is commonly used, but costs >\$200,000 USD [\[2\]](#)!



# Triple Modular Redundancy



# Triple Modular Redundancy

TMR can be added manually...

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

Can we automate it?

# TaMaRa methodology

---

Implement TMR as a pass in an EDA synthesis tool.

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

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.

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.

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 [\[8\]](#)

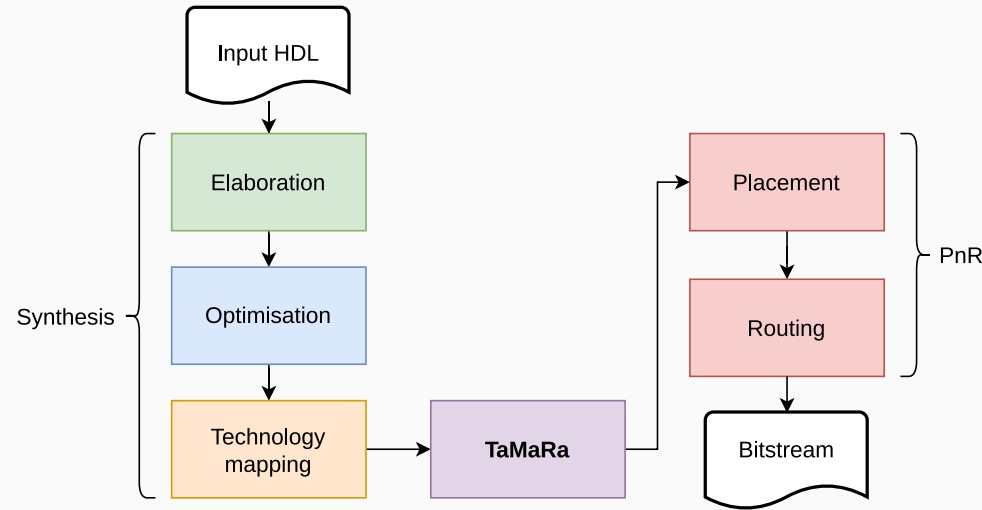


# The TaMaRa algorithm

TaMaRa will mainly be netlist-driven, using Johnson's [6] voter insertion algorithm.

Also aim to propagate a (\* triplicate \*) HDL annotation to select TMR granularity (similar to Kulis [4]).

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



# The TaMaRa algorithm

Why netlist driven with the `(* triplicate *)` annotation?

# The TaMaRa algorithm

Why netlist driven with the `(* triplicate *)` annotation?

- Removes the possibility of Yosys optimisation eliminating redundant TMR logic
- Removes the necessity of complex blackboxing logic and trickery to bypass the normal design flow
- Cell type shouldn't matter, TaMaRa targets FPGAs and ASICs
- Still allows selecting TMR granularity - **best of both worlds**

Comprehensive verification procedure using formal methods, simulation and fuzzing.

Driven by SymbiYosys tools *eqy* and *mcy*

- In turn driven by theorem provers/SAT solvers

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

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.

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 SEUs (single bit only)

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 SEUs (single bit only)

- Ensures TaMaRa does its job!



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 SEUs (single bit only)

- Ensures TaMaRa does its job!

Also considering Beltrame's verification tool [\[9\]](#), and other literature on TMR formal verification.

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

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

Idea:

1. Use Verismith [\[10\]](#) 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 [\[10\]](#) 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

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

Idea:

1. Use Verismith [\[10\]](#) 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

- We need valid testbenches for these random circuits, how would we generate that?
- Under active research in academia (may not be possible at the moment)

We want to simulate an SEU environment.

- UQ doesn't have the capability to expose FPGAs to real radiation
- Physical verification is challenging (how do you measure it?)

We want to simulate an SEU environment.

- UQ doesn't have the capability to expose FPGAs to real radiation
- Physical verification is challenging (how do you measure it?)

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

- Each simulator has different trade-offs
- Currently considering picorv32 or Hazard3 as the DUT
- Most likely will use Verilator or cxxrtl

We want to simulate an SEU environment.

- UQ doesn't have the capability to expose FPGAs to real radiation
- Physical verification is challenging (how do you measure it?)

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

- Each simulator has different trade-offs
- Currently considering picorv32 or Hazard3 as the DUT
- Most likely will use Verilator or cxxrtl

Concept:

- Iterate over the netlist, randomly consider flipping a bit every cycle
- Write a self-checking testbench and ensure that the DUT responds correctly



# Technical implementation

Implemented in C++20, using CMake.

Load into Yosys: `plugin -i libtamara.so`

TMR is implemented as two separate commands: `tamara_propagate` and `tamara_tmr`

# Technical implementation

Implemented in C++20, using CMake.

Load into Yosys: `plugin -i libtamara.so`

TMR is implemented as two separate commands: `tamara_propagate` and `tamara_tmr`

Run `tamara_propagate` after `read_verilog` to propagate the `(* triplicate *)` annotations.

# Technical implementation

Implemented in C++20, using CMake.

Load into Yosys: `plugin -i libtamara.so`

TMR is implemented as two separate commands: `tamara_propagate` and `tamara_tmr`

Run `tamara_propagate` after `read_verilog` to propagate the `(* triplicate *)` annotations.

Run `tamara_tmr` after techmapping to perform triplication and voter insertion (add TMR).

## Current status & future

---

# Current status

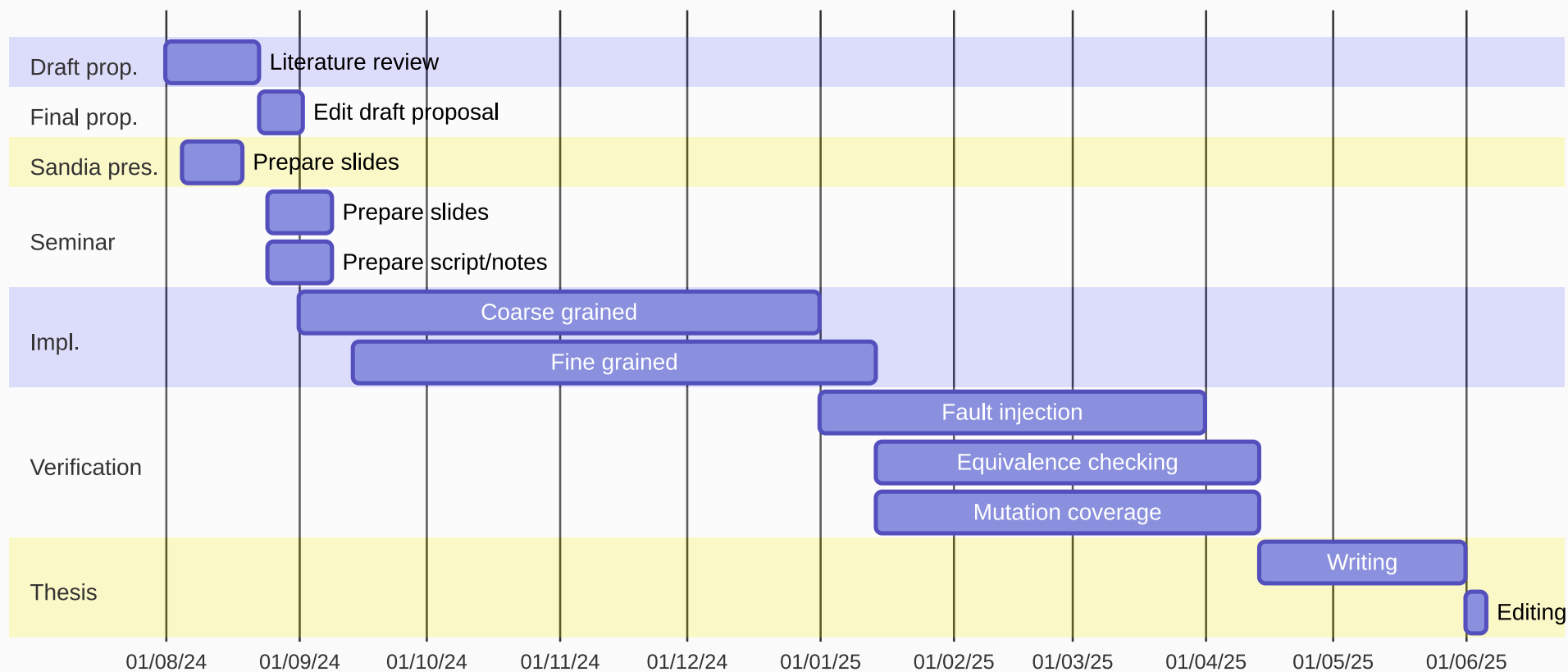
Mostly initial academic work: literature reviews, project proposals, etc.

`tamara_propagate` pass already basically complete (just under 100 lines of C++).

Work for the next ~6 months will focus on understanding, implementing and verifying Johnson's [6] algorithm in Yosys, and exploring Beltrame's [9] verification.

# The future

## TaMaRa Project Plan



This will be implemented for my Honours thesis over the next 1 year.

- Honours is kind of like mini masters, it's an Australia-specific thing
- Supervised by Assoc. Prof. John Williams (former PetaLogix, Xilinx)
- Ideally would like to publish if all goes well

This will be implemented for my Honours thesis over the next 1 year.

- Honours is kind of like mini masters, it's an Australia-specific thing
- Supervised by Assoc. Prof. John Williams (former PetaLogix, Xilinx)
- Ideally would like to publish if all goes well

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



# The future

This will be implemented for my Honours thesis over the next 1 year.

- Honours is kind of like mini masters, it's an Australia-specific thing
- Supervised by Assoc. Prof. John Williams (former PetaLogix, Xilinx)
- Ideally would like to publish if all goes well

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

Ideally, TaMaRa will be released open-source under MPL 2.0.

- Pending university IP shenanigans...

# 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
- Netlist-driven algorithm based on Johnson's work [\[6\]](#)
- **Key goal:** "Click a button" and have any circuit run in space/in high reliability environments!

# 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](https://doi.org/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-austrochip2013.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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/10.7717/peerj-cs.21).
- [10] 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](https://doi.org/10.1145/3373087.3375310).

Thank you! Any questions?