An automated triple modular redundancy EDA flow for Yosys

REIT4882 Thesis Draft Project Proposal

Matt Young

46972495

m.young2@student.uq.edu.au

August 2024

Contents

1. Introduction

For safety-critical sectors such as aerospace and defence, both ASICs and FPGA gateware must be designed to be fault tolerant to prevent catastrophic malfunctions. In the context of digital electronics, *fault tolerant* means that the design is able to gracefully recover and continue operating in the event of a fault, or upset. A Single Event Upset (SEU) occurs when ionising radiation strikes a transistor on a digital circuit, causing it to transition from a 1 to a 0, or vice versa. This type of upset is most common in space, where the Earth's atmosphere is not present to dissipate the ionising particles [1]. On an unprotected system, an unlucky SEU may corrupt the system's state to such a severe degree that it may cause destruction or loss of life - particularly important given the safety-critical nature of most space-fairing systems (satellites, crew capsules, missiles, etc). Thus, fault tolerant computing is widely studied and applied for space-based computing systems.

One common fault-tolerant design technique is Triple Modular Redundancy (TMR), which mitigates SEUs by triplicating key parts of the design and using voter circuits to select a non-corrupted result if an SEU occurs. Typically, TMR is manually designed at the Hardware Description Language (HDL) level, for example, by manually instantiating three copies of the target module, designing a voter circuit, and linking them all together. However, this approach is an additional time-consuming and potentially error-prone step in the already complex design pipeline.

TODO: diagram of TMR

To address these issues, I propose TaMaRa: a novel fully automated TMR flow for the open source Yosys EDA tool [2].

Modern digital ICs and FPGAs are described using Hardware Description Languages (HDLs), such as SystemVerilog or VHDL. The process of transforming this high level description into a photolithography mask (for ICs) or bitstream (for FPGAs) is achieved through the use of Electronic Design Automation (EDA) tools. This generally comprises of the following stages:

- **Synthesis**: The transformation of a high-level textual HDL description into a lower level synthesisable netlist.
  - **Elaboration:** Includes the instantiation of HDL modules, resolution of generic parameters and constants. Like compilers, synthesis tools are typically split into frontend/backend, and elaboration could be considered a frontend/language parsing task.
  - **Optimisation:** This includes a multitude of tasks, anywhere from small peephole optimisations, to completely re-coding FSMs. In commercial tools, this is typically timing driven.
  - **Technology mapping:** This involves mapping the technology-independent netlist to the target platform, whether that be FPGA LUTs, or ASIC standard cells.
- **Placement**: The process of optimally placing the netlist onto the target device. For FPGAs, this involves choosing which logic elements to use. For digital ICs, this is much more complex and manual - usually done by dedicated layout engineers who design a *floorplan*.
- **Routing**: The process of optimally connecting all the placed logic elements (FPGAs) or standard cells (ICs).

Due to their enormous complexity and cutting-edge nature, most IC EDA tools are commercial proprietary software sold by the big three vendors: Synopsys, Cadence and Siemens. These are economically infeasible for almost all researchers, and even if they could be licenced, would not be possible to extend to implement custom synthesis passes. The major FPGA vendors, AMD and Intel, also develop their own EDA tools for each of their own devices, which are often cheaper or free. However, these tools are still proprietary software and cannot be modified by researchers. Until recently, there was no freely available, research-grade, open-source EDA tool available for study and improvement. That changed with the introduction of Yosys [2]. Yosys is a capable synthesis tool that can emit optimised netlists for various FPGA families as well as a few silicon process nodes (e.g. Skywater 130nm). Importantly, for this thesis, Yosys can be modified either by changing the source code or by developing modular plugins that can be dynamically loaded at runtime. Due to specific advice from the Yosys development team [3], TaMaRa will be developed as a loadable C++ plugin.

TODO diagram of the synthesis flow

2. Aims

This thesis is governed by two overarching aims:

- To design a C++ plugin for the Yosys synthesis tool that, when presented with any Yosys-compatible HDL input, will apply an algorithm to turn the selected HDL module(s) into a triply modular redundant design, suitable for space.
- To design and a implement a comprehensive verification process for the above pass, including the use of formal methods, HDL simulation, fuzzing and potential real-life radiation exposure.

Much like designing a pass for a compiler, designing a pass for an EDA tool is no light undertaking. It needs to handle all possible designs the user may provide as input, and provide a high degree of assurance of correctness. This is particularly important given the safety-critical nature of the designs users may provide to TaMaRa. I do not undertake this lightly, and the rigorous verification methodology is a necessity to produce a pass worth using.

These two major aims can be broken down into smaller aims. Under the design pipeline:

- Research the applications of graph theory to TODO

2.1. Engineering requirements

Due to the large and complex nature of the TaMaRa development process, I decided it beneficial to apply the MoSCoW engineering requirements system. I present the requirements and their justifications. The capitalised keywords are to be interpreted according to RFC 2119 [4].

TMR pass requirements

Requirement	Justification
Tamara SHALL be implemented as a C++ pass for the Yosys synthesis tool	Yosys is certainly going to be the synthesis tool used, and the C++ plugin API is the most stable.
Tamara SHALL process the design in such a way that triple modular redundancy (TMR) is applied to the selected HDL module(s), protecting it from SEUs	This is the overarching goal of the thesis.
Tamara MAY operate at any desired level of granularity - anywhere from RTL code level, to elaboration, to techmapping - but it SHALL operate on at least one level of granularity	As long as the TMR is implemented correctly, it doesn't matter what level of granularity the algorithm uses. Each level of granularity has different trade-offs which still require research at this stage.
Tamara SHOULD compare coarse and fine grained TMR	It would be interesting to see the area and reliability effects of applying TMR in at least two different ways. This is left as a SHOULD in case of serious time constraints.
Tamara SHOULD be capable of handling large designs, up to and including picorv32, in reasonable amounts of time and memory	Also supports the overarching goal of the thesis, but left as a SHOULD in case of major unforeseen implementation issues with the performance.
Tamara MAY handle FPGA primitives like SRAMs and DSP slices	Most likely will not handle these primitives as there's no reliable way to replicate them across all FPGA vendors supported by Yosys.
Tamara MAY make the voters themselves redundant	Could be added for extra assurance, but not typically considered necessary in industry.
Tamara SHOULD NOT be timing driven	Timing is best left up to the P&R tool (Nextpnr). Although some EDA synthesis tools are timing driven, Yosys currently is not.
Tamara SHOULD have a clean codebase through the use of tools like clang-tidy	Easy to implement and highly desirable but not strictly necessary for correct functioning.
Tamara SHALL NOT consider multi-bit upsets	Although multi-bit upsets may occur in practice, this work focuses on SEUs in particular. MBUs are much less likely (TODO citation?) and require significant area increases due to extra voters (TODO citation?)

Verification requirements

Requirement	Justification
Verification simulation SHALL be performed using one or more of: Verilator, Icarus Verilog, cxxrtl	These are the best open-source simulation tools, and each have different trade-offs (e.g. Verilator is fast, but not sub-cycle accurate).
Verification SHOULD involve a complex design (e.g. picorv32 CPU) in a simulated SEU environment	This is an important final test, but is left as a SHOULD requirement in case of major unforeseen issues applying TMR to large designs.
Verification SHALL involve equivalence checking (formally proving that a design acts the same before and after TMR) using *SymbiYosys* and *eqy*	Equivalence checking is necessary to formally prove that the TMR pass does not modify the behaviour of the design, only that it adds TMR.
Verification MAY involve fuzzing equivalence checking (generating random RTL modules, applying TMR, and checking they're identical)	It's not clear at the time of writing whether a fully end-to-end, automated fuzzing approach for equivalence checking is possible.
Verification SHALL involve mutation coverage (injecting faults into the design and formally proving that TaMaRa mitigates them) using *mcy*	Mutation coverage is necessary to formally prove that the TMR pass correctly mitigates SEUs.
Verification MAY involve fuzzing mutation coverage, if such a thing is possible	Early research indicates that the generation of random RTL *as well as* random testbenches is still under active research in academia.
Verification MAY involve a physical, real-life radiation test whereby an FPGA with a Tamara bitstream on it is exposed to radiation	It's not known at the time of writing whether UQ has the facilities to perform this test, or whether the risks caused by radiation exposure are worth the investigation.

3. Literature review

Although the concept of N-modular redundancy dates back to antiquity, the application of triple modular redundancy to computer systems was first introduced in academia by R. Lyons and W. Vanderkul [5]. Like much of computer science, however, the authors trace the original concept back to John von Neumann. In addition to introducing the application of TMR to computer systems, the authors also provide a rigorous Monte-Carlo mathematical analysis of the reliability of TMR. One important takeaway from this is that the only way to make a system reliably redundant is to split it into multiple components, each of which is more reliable than the system as a whole. In the modern FPGA concept, this implies applying TMR at an RTL module level, although as we will soon see, more optimal and finer grained TMR can be applied. Although their Monte Carlo analysis shows that TMR dramatically improves reliability, they importantly show that as the number of modules M in the computer system increases, the computer will eventually become less reliable. This is due to the fact that the voter circuits may not themselves be perfectly reliable, and is important to note for FPGA and ASIC designs which may instantiate hundreds or potentially thousands of modules.

TODO more literature

Recognising that prior literature focused mostly around manual or theoretical TMR, and the limitations of a manual approach, J. M. Johnson and M. J. Wirthlin [6] introduced four algorithms for the automatic insertion of TMR voters in a circuit, with a particular focus on timing and area trade-offs. Together with the thesis this paper was based on [7], these two publications form the seminal works on automated TMR.

Whilst they provide an excellent design of TMR insertion algorithms, and a very thorough analysis of their area and timing trade-offs, J. M. Johnson and M. J. Wirthlin [6] do not have a rigorous analysis of the correctness of these algorithms. They produce experimental results demonstrating the timing and area trade-offs of the TMR algorithms on a real Xilinx Virtix 1000 FPGA, up to the point of P&R, but do not run it on a real device. More importantly, they also do not have any formal verification or simulated SEU scenarios to prove that the algorithms both work as intended, and keep the underlying behaviour of the circuit the same. Finally, in their thesis [7], the authors state that the benchmark designs were synthesised using a combination of the commercial Synopsys Synplify tool, and the *BYU-LANL Triple Modular Redundancy (BL-TMR) Tool*. This Java-based set of tools ingest EDIF-format netlists, perform TMR on them, and write the processed result to a new EDIF netlist, which can be re-ingested by the synthesis program for place and route. This is quite a complex process, and was also designed before Yosys was introduced in 2013. It would be very beneficial if the TMR pass was instead integrated directly into the synthesis tool - which is only possible for Yosys, as Synplify is commercial proprietary software. This is especially important for industry users who often have long and complicated synthesis flows.

It's also worth noting that D. Skouson, A. Keller, and M. Wirthlin [8] introduced SpyDrNet, a Python-based netlist transformation tool that also implements TMR using the same algorithm as above. SpyDrNet is a great general purpose transformation tool for research purposes, but again is a separate tool that is not integrated *directly* into the synthesis process. We instead aim to make a *production* ready tool, with a focus on ease-of-use, correctness and performance.

4. Timeline

TODO

References

[1] M. O'Bryan, "Single Event Effects." Accessed: Jul. 29, 2024. [Online]. Available: https://radhome.gsfc.nasa.gov/radhome/see.htm

[2] C. Wolf and J. Glaser, "Yosys - A Free Verilog Synthesis Suite," in *Proceedings of Austrochip 2013*, 2013.

[3] N. Engelhardt, May 2024.

[4] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," Mar. 1997. [Online]. Available: https://www.ietf.org/rfc/rfc2119.txt

[5] R. Lyons and W. Vanderkul, "The Use of Triple-Modular Redundancy to Improve Computer Reliability," *IBM Journal of Research and Development*, vol. 6, pp. 200–209, 1962.

[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] J. Johnson, J. Mark, M. Wirthlin, B. Hutchings, and B. Nelson, "Synchronization Voter Insertion Algorithms for FPGA Designs Using Triple Modular Redundancy," 2010.

[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.