An automated triple modular redundancy EDA flow for Yosys

REIT4882 Draft Thesis Project Proposal

Matt Young

School of Electrical Engineering and Computer Science

University of Queensland

August 2024

Abstract

Safety-critical sectors require Application Specific Integrated Circuit (ASIC) designs and Field Programmable Gate Array (FPGA) gateware to be fault-tolerant. In particular, space-fairing computers need to mitigate the effects of Single Event Upsets (SEUs) caused by ionising radiation. 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. Typically, this is manually implemented by designers at the Hardware Description Language (HDL) level, but this is error-prone and time-consuming. Leveraging the power and flexibility of the open-source Yosys Electronic Design Automation (EDA) tool, in this document I will propose TaMaRa: a novel fully automated TMR flow, implemented as a Yosys plugin. I provide a comprehensive review of relevant automated TMR literature, and provide a detailed plan of the TaMaRa project, including its design and verification.

Contents

1. Background and introduction

For safety-critical sectors such as aerospace and defence, both Application Specific Integrated Circuits (ASICs) and Field Programmable Gate Array (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 magnetosphere 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 (see Figure 1). 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.

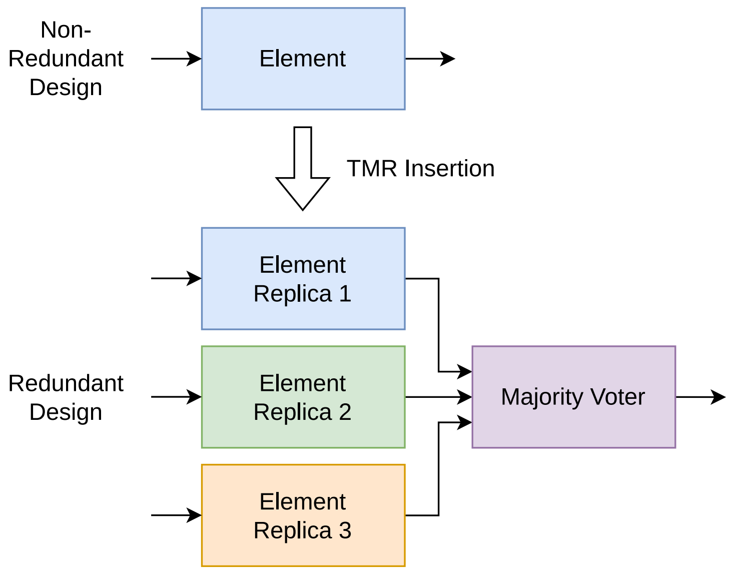


Figure 1: Diagram demonstrating how TMR is inserted into an abstract design

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 (Figure 2):

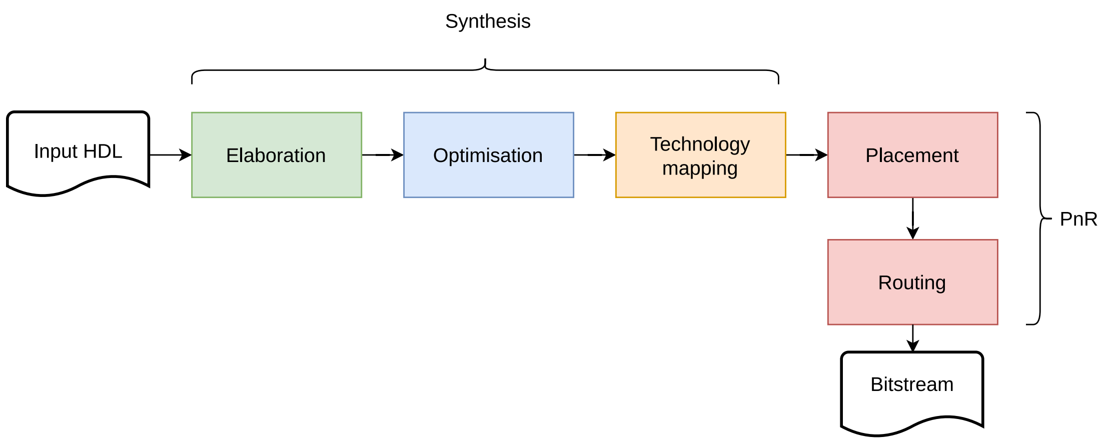


Figure 2: Simplified representation of a typical EDA synthesis flow

- **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). When combined with the Nextpnr place and route tool [3], Yosys+nextpnr forms a fully end-to-end FPGA synthesis flow for Lattice iCE40 and ECP5 devices. 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.

2. Literature review

2.1. Introduction, methodology and terminology

The automation of triple modular redundancy, as well as associated topics such as general fault-tolerant computing methods and the effects of SEUs on digital hardware, have been studied a fair amount in academic literature. Several authors have invented a number of approaches to automate TMR, at various levels of granularity, and at various points in the FPGA/ASIC synthesis pipeline. This presents an interesting challenge to systematically review and categorise. To address this, I propose that all automated TMR approaches can be fundamentally categorised into the following dichotomy:

- **Design-level approaches** ("thinking in terms of HDL"): These approaches treat the design as *modules*, and introduce TMR by replicating these modules. A module could be anything from an entire CPU, to a register file, to even a single combinatorial circuit or AND gate. Once the modules are replicated, voters are inserted.
- **Netlist-level approaches** ("thinking in terms of circuits"): These approaches treat the design as a *circuit* or *netlist*, which is internally represented as a graph. TMR is introduced using graph theory algorithms to *cut* the graph in a particular way and insert voters.

Using these two design paradigms as a guiding point, I analyse the literature on automated TMR, as well as background literature on fault-tolerant computing.

2.2. Fault tolerant computing and redundancy

The application of triple modular redundancy to computer systems was first introduced into academia by Lyons and Vanderkul [4] in 1962. 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 Register Transfer Level (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.

For ASICs, instead of triple modular redundancy, ASICs can be designed using rad-hardened CMOS process nodes or design techniques. Much has been written about rad-hardened microprocessors, of which many are deployed (and continue to be deployed) in space to this day. One such example is the RAD750 [5], a rad-hardened PowerPC CPU for space applications designed by Berger et al. of BAE Systems. They claim "5-6 orders of magnitude" better SEU performance compared to a stock PowerPC 750 under intense radiation conditions. The processor is manufactured on a six-layer commercial 250 nm process node, using specialty design considerations for the RAM, PLLs, and standard cell libraries. Despite using special design techniques, the process node itself is standard commercial CMOS node and is not inherently rad-hardened. The authors particularly draw attention to the development of a special SEU-hardened RAM cell, although unfortunately they do not elaborate on the exact implementation method used. However, they do mention that these techniques increase the die area from 67 mm^2 in a standard PowerPC 750, to 120 mm^2 in the RAD750, a 1.7x increase. Berger et al. also used an extensive verification methodology, including the formal verification of the gate-level netlist and functional VHDL simulation. The RAD750 has been deployed on numerous high-profile missions including the James Webb Space Telescope and Curiosity Mars rover. Despite its wide utilisation, however, the RAD750 remains extremely expensive - costing over $200,000 USD in 2021 [6]. This makes it well out of the reach of research groups, and possibly even difficult to acquire for space agencies like NASA.

In addition to commercial CMOS process nodes, there are also specialty rad-hardened process nodes designed by several fabs. One such example is Skywater Technologies' 90 nm FD-SOI ("Fully Depleted Silicon-On-Insulator") node. The FD-SOI process, in particular, has been shown to have inherent resistance to SEUs and ionising radiation due to its top thin silicon film and buried insulating oxide layer [7]. Despite this, unfortunately, FD-SOI is an advanced process node that is often expensive to manufacture.

Instead of the above, with a sufficiently reliable TMR technique (that this research ideally would like to help create), it should theoretically be possible to use a commercial-off-the-shelf (COTS) FPGA for mission critical space systems, reducing cost enormously - this is one of the key value propositions of automated TMR research. Of course, TMR is not flawless: its well-known limitations in power, performance and area (PPA) have been documented extensively in the literature, particularly by Johnson [8], [9]. Despite this, TMR does have the advantage of being more general purpose and cost-effective than a specially designed ASIC like the RAD750. TMR can be applied to any design, FPGA or ASIC, at various different levels of granularity and hierarchy, allowing for studies of different trade-offs. For ASICs in particular, unlike the RAD750, TMR as a design technique does not need to be specially ported to new process nodes: an automated TMR approach could be applied to a circuit on a 250 nm or 25 nm process node without any major design changes. Nonetheless, specialty rad-hardened ASICs will likely to see future use in space applications. In fact, it's entirely possible that a rad-hardened FPGA *in combination* with an automated TMR technique is the best way of ensuring reliability.

**TODO more background literature on other approaches to rad-hardening: rad-hardened CMOS and TMR CPUs and
scrubbing**

TODO more literature defining probabilities and effects of SEUs on ASICs/FPGAs in space

2.3. Netlist-level approaches

Recognising that prior literature focused mostly around manual or theoretical TMR, and the limitations of a manual approach, Johnson and Wirthlin [8] 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 [9], these two publications form the seminal works on automated TMR for digital EDA. **TODO more details on Johnson**

Whilst they provide an excellent design of TMR insertion algorithms, and a very thorough analysis of their area and timing trade-offs, Johnson and Wirthlin 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 his thesis [9], Johnson states 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 better 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.

Later, Skouson et al. [10] (from the same lab as above) 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. I instead aim to make a *production* ready tool, with a focus on ease-of-use, correctness and performance.

Using a similar approach, Benites and Kastensmidt [11], and Benites' thesis [12], introduce an automated TMR approach implemented as a Tcl script for use in Cadence tools. They distinguish between "coarse grained TMR" (which they call "CGTMR"), applied at the RTL module level, and "fine grained TMR" (which they call "FGTMR"), applied at the sub-module (i.e. net) level. Building on that, they develop an approach that replicates both combinatorial and sequential circuits, which they call "fine grain distributed TMR" or "FGDTMR". They split their TMR pipeline into three stages: implementation ("TMRi"), optimisation ("TMRo"), and verification ("TMRv"). The implementation stage works by creating a new design to house the TMR design (which I'll call the "container design"), and instantiating copies of the original circuit in the container design. Depending on which mode the user selects, the authors state that either each "sequential gate" will be replaced by three copies and a voter, or "triplicated voters" will be inserted. What happens in the optimisation stage is not clear as Benites does not elaborate at all, but he does state it's only relevant for ASICs and involves "gate sizing". For verification, Benites uses a combination of fault-injection simulation (where SEUs are intentionally injected into the simulation), and formal verification through equivalence checking. Equivalence checking involves the use of Boolean satisfiability solvers ("SAT solvers") to mathematically prove one circuit is equivalent to another. Benites' key verification contribution is identifying a more optimal way to use equivalence checking to verify fine-grained TMR. He identified that each combinatorial logic path will be composed of a path identical to the original logic, plus one or more voters. This way, he only has to prove that each "logic cone" as he describes it is equivalent to the original circuit. Later on, he also uses a more broad-phase equivalence checking system to prove that the circuits pre and post-TMR have the same behaviours.

One of the most important takeaways from these works are related to clock synchronisation. Benites interestingly chooses to not replicate clocks or asynchronous reset lines, which he states is due to clock skew and challenges with multiple clock domains created by the redundancy. Due to the clear challenges involved, ignoring clocks and asynchronous resets is a reasonable limitation introduced by the authors, and potentially reasonable for us to introduce as well. Nonetheless, it is a limitation I would like to address in TaMaRa if possible, since leaving these elements unprotected creates a serious hole that would likely preclude its real-world usage[1]. Arguably, the most important takeaway from Benites' work is the use of equivalence checking in the TMR verification stage. This is especially important since Johnson [8] did not formally verify his approach. Benites' usage of formal verification, in particular, equivalence checking, is an excellent starting point to design the verification methodology for TaMaRa.

On the lower level side, Hindman et al. [13] introduce an ASIC standard-cell based automated TMR approach. When digital circuits are synthesised into ASICs, they are technology mapped onto standard cells provided by the foundry as part of their Process Design Kit (PDK). For example, SkyWater Technology provides an open-source 130 nm ASIC PDK, which contains standard cells for NAND gates, muxes and more [14]. The authors design a TMR flip-flop cell, known as a "Triple Redundant Self Correcting Master-Slave Flip-Flop" (TRSCMSFF), that mitigates SEUs at the implementation level. Since this is so low level and operates below the entire synthesis/place and

[1]My view is essentially that an unprotected circuit remains unprotected, regardless of how difficult it is to correct clock skewing. In other words, simply saying that the clock skew exists doesn't magically resolve the issue. In Honours, we are severely time limited, but it's still my goal to address this limitation if possible.

route pipeline, their approach has the advantage that *any* design - including proprietary encrypted IP cores that are (unfortunately) common in industry - can be made redundant. Very importantly, the original design need not be aware of the TMR implementation, so this approach fulfills my goal of making TMR available seamlessly to designers. The authors demonstrate that the TRSCMSFF cell adds minimal overhead to logic speed and power consumption, and even perform a real-life radiation test under a high energy ion beam. Overall, this is an excellent approach for ASICs. However, this approach, being standard-cell specific, cannot be applied to FPGA designs. Rather, the FPGA manufacturers themselves would need to apply this method to make a series of specially rad-hardened devices. It would also appear that designers would have to port the TRSCMSFF cell to each fab and process node they intend to target. While TaMaRa will have worse power, performance and area (PPA) trade-offs on ASICs than this method, it is also more general in that it can target FPGAs *and* ASICs due to being integrated directly into Yosys. Nevertheless, it would appear that for the specific case of targeting the best PPA trade-offs for TMR on ASICs, the approach described in [13] is the most optimal one available.

TODO Xilinx industry paper

2.4. Design-level approaches

Several authors have investigated applying TMR directly to HDL source code. One of the most notable examples was introduced by Kulis [15], through a tool he calls “TMRG”. TMRG operates on Verilog RTL by implementing the majority of a Verilog parser and elaborator from scratch. It takes as input Verilog RTL, as well as a selection of Verilog source comments that act as annotations to guide the tool on its behaviour. In turn, the tool modifies the design code and outputs processed Verilog RTL that implements TMR, as well as Synopsys Design Compiler design constraints. Like the goal of TaMaRa, the TMRG approach is designed to target both FPGAs and ASICs, and for FPGAs, Kulis correctly identifies the issue that not all FPGA blocks can be replicated. For example, a design that instantiates a PLL clock divider on an FPGA that only contains one PLL could not be replicated. Kulis also correctly identifies that optimisation-driven synthesis tools such as Yosys and Synopsys DC will eliminate TMR logic as part of the synthesis pipeline, as the redundancy is, by nature, redundant and subject to removal. In Yosys, this occurs specifically in the `opt_share` and `opt_clean` passes according to specific advice from the development team [16]. However, unlike Synopsys DC, Yosys is not constraint driven, which means that Kulis’ constraint-based approach to preserving TMR logic through optimisation would not work in this case. Finally, since TMRG re-implements the majority of a synthesis tool’s frontend (including the parser and elaborator), it is limited to only supporting Verilog. Yosys natively supports Verilog and some SystemVerilog, with plugins [17] providing more complete SV and VHDL support. Since TaMaRa uses Yosys’ existing frontend, it should be more reliable and useable with many more HDLs.

Lee et al. [18] present “TLegUp”, an extension to the prior “LegUp” High Level Synthesis (HLS) tool. As stated earlier in this document, modern FPGAs and ASICs are typically designed using Hardware Description Languages (HDLs). HLS is an alternative approach that aims to synthesise FPGA/ASIC designs from high-level software source code, typically the C or C++ programming languages. On the background of TMR in FPGAs in general, the authors identify the necessity of “configuration scrubbing”, that is, the FPGA reconfiguring itself when one of the TMR voters detects a fault. Neither their TLegUp nor our TaMaRa will address this directly, instead, it’s usually best left up to the FPGA vendors themselves (additionally, TaMaRa targets ASICs which cannot be runtime reconfigured). Using voter insertion algorithms inspired by Johnson [8], the authors operate on LLVM Intermediate Representation (IR) code generated by the Clang compiler. By inserting voters before both the HLS and RTL synthesis processes have been run, cleverly the LegUp HLS system will account for the critical path delays introduced by the TMR process. This means that, in addition to performance benefits, pipelined TMR voters can be inserted. The authors also identify four major locations to insert voters: feedback paths from the datapath, FSMs, memory output signals and output ports on the top module. Although TaMaRa isn't HLS-based, Yosys does have the ability to detect abstract features like FSMs, so we could potentially follow this methodology as well. The authors also perform functional simulation using Xilinx ISE, and a real-world simulation by using a Microblaze soft core to inject faults into various designs. They state TLegUp reduces the error rate by 9x, which could be further improved by using better placement algorithms. Despite the productivity gains, and in this case the benefits of pipelined voters, HLS does not come without its own issues. Lahti et al. [19] note that the quality of HLS results continues to be worse than those designed with RTL, and HLS generally does not see widespread industry usage in production designs. One other key limitation that Lee et al. do not fully address is the synthesis optimisation process incorrectly removing the redundant TMR logic. Their workaround is to disable "resource sharing options in Xilinx ISE to prevent sub-expression elimination", but ideally we would not have to disable such a critical optimisation step just to implement TMR. TaMaRa aims to address this limitation by working with Yosys directly.

Khatri et al. [20] propose a similar, albeit much less sophisticated approach. They develop a Matlab script that ingests a Verilog RTL module, instantiates it three times (to replicate it), and wraps it in a new top-level module. They also propose a new majority voter using a 2:1 multiplexer, which they claim has 50% better Fault Mask Ratio (FMR) than the traditional AND-gate based approach. The authors test only one single circuit, described as a "simple benchmark design", in a fault-injection RTL simulation. They do not use any systematic verification methodology that other authors use, only injecting a limited number of faults into one single design. Khatri et al. also make no mention of the limitations that other authors identify for this high-level TMR approach. This includes the PPA trade-offs that Benites [11] and Johnson [8] identify, as well as the logic optimisation and resource utilisation issues that Kulis [15] correctly pointed out. The decision to develop the tool as a GUI application is highly questionable as it significantly interrupts the typical command-line based synthesis flow. This is especially true for ASICs, which have very long synthesis pipelines that are typically Tcl scripted. Whilst this approach is not exactly very thorough or high quality, the higher reliability voter circuit may possibly be worth investigating.

2.5. TMR verification

While Benites [11], [12] discusses verification of the automated TMR process, and other authors [18], [20], [13], [5] also use various different verification/testing methodologies, there is also some literature that exclusively focuses on the verification aspect. Verification is one of the most important parts of this process due to the safety-critical nature of the devices TMR is typically deployed to. Additionally, there are interesting trade-offs between different verification methodologies, particularly fault injection vs. formal verification.

Beltrame [21] uses a divide and conquer approach for TMR netlist verification. Specifically, identifying limitations with prior fault-injection simulation and formal verification techniques, he presents an approach described as fault injection combined with formal verification: instead of simulating the entire netlist with timing accurate simulation, he uses a behavioural timeless simulation of small submodules ("logic cones") extracted by automatic analysis. This seems to be an effective and rigorous approach, and the code for the tool appears to be available on GitHub as "InFault". It would be highly worthwhile investigating the use of this tool for verification, as it has already been proven in prior research and may overall save time. That being said, a quick analysis of the code appears to reveal it to be "research quality" (i.e. zero documentation and seems partially unfinished). The question would be whether figuring out how to use InFault takes more time than simply implementing formal verification ourselves in Yosys.

TODO more on Beltrame

TODO other verification papers?

3. Project plan

3.1. Aims of the project

The development of TaMaRa aims to answer the following research questions:

1. What are the trade-offs between low-level/high-level TMR; and TMR in general?
   - What are the power, performance and area (PPA) trade-offs?
   - What are the reliability trade-offs?
   - Is coarse-grained (high-level) or fine-grained (low-level) TMR better?
   - What is the best point in the synthesis pipeline to perform TMR?
2. What are the best methods to verify TMR, and what are the trade-offs between them?
   - Is formal verification possible?
   - Does fault-injection simulation prove the validity of TMR to a sufficient level?
3. Can an automated TMR approach be implemented to production-grade quality? (i.e., can it accept all circuits and generate a reasonable output in a reasonable amount of time?)

To investigate these research questions, I propose two major project aims we should set out to achieve:

1. 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.
2. 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.

To further analyse the situation, I decided to break up each major goal into further sub-goals.

Major aim 1 sub-goals:
- Design and implement an algorithm for doing netlist-level TMR in Yosys
- Research the applications of graph theory, including connected components, to automated netlist-level TMR
- Understand and decide between the different trade-offs of where in the synthesis pipeline TMR should be implemented
- Understand and consider the effects of clock skew and clock domain crossing for clock lines on TMR circuits

Major aim 2 sub-goals:
- Understand the application of equivalence checking (Yosys *eqy*) to circuits
- Understand the application of formal fault checking/coverage (Yosys *mcy*) to circuits
- Design and implement an RTL simulation with fault injection for the purpose of verifying TMR
- Understand the limitations and benefits of formal verification vs. RTL simulation for the purposes of verifying TMR

3.2. 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, based on the goals presented above. I present the requirements and their justifications. The capitalised keywords are to be interpreted according to RFC 2119 [22].

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 to occur [23] and requires more complex methods to deal with [24].

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.

Requirement	**Justification**
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 SHOULD NOT involve a physical, real-life radiation test whereby an FPGA with a TaMaRa bitstream on it is exposed to radiation	UQ does not have the facilities to expose a real-life FPGA to radiation. Even if it did, the risks and challenges created by this verification approach would not be worth its utilisation.

3.3. Implementation plan

To design the TaMaRa algorithm, I synthesise existing approaches from the literature review to form a novel approach suitable for implementation in Yosys. Specifically, I synthesise the voter insertion algorithms of Johnson [8], the RTL annotation-driven approach of Kulis [15], and parts of the verification methodology of Benites [11], to form the TaMaRa algorithm and verification methodology. Based on the dichotomy identified in Section 2.1, TaMaRa will be classified as a *netlist-level* approach, as the algorithms are designed by treating the design as a circuit (rather than HDL). Despite this, TaMaRa aims to combine the best of both worlds, by supporting a `triplicate` annotation that allows end users to select the level of granularity they want to apply TMR at. Johnson's algorithm processes the entire netlist in one go, whereas TaMaRa aims to use the `triplicate` annotation to allow more fine-grained selections, allowing studies to compare different levels of TMR design granularity.

I propose a modification to the synthesis flow that inserts TaMaRa after technology mapping (Figure 3). This means that the circuit can be processed at a low level, without having to worry about optimisation removing the redundant logic.

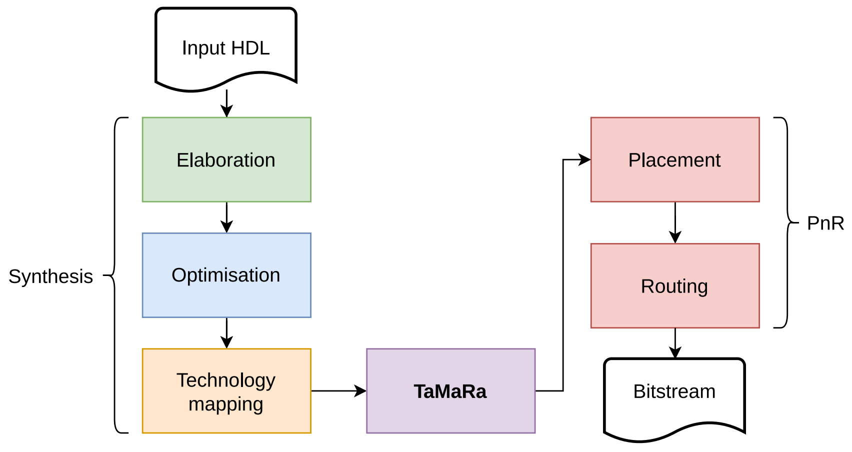


Figure 3: Proposed modification to synthesis flow including TaMaRa TMR

For the specific implementation details, I propose to implement TaMaRa as a Yosys plugin in C++20, using CMake as the build tool. This will compile a Linux shared library, `libtamara.so`, which can be loaded into Yosys using the command `plugin -i libtamara.so`. Then, various commands (yet to be determined but likely `tamara` and possibly `tamara_propagate`) will be made available to end users. Since Yosys is open source, there are two ways to add changes: either contributing improvements to the upstream codebase, or by writing a plugin. I am electing to develop a plugin due to specific personal advice from the Yosys development team [16].

Briefly, the general overview of the TaMaRa algorithm looks as follows:

1. Propagate the `(* triplicate *)` annotation. For example, if a module is marked with triplicate, then all ports and processes in it should also be marked with `triplicate`.
2. Mark cells with the `triplicate` annotation in the netlist
3. Replicate each cell marked with the `triplicate` annotation an additional 2 times (so there are now 3 instances of each cell with the `triplicate` annotation)
4. Wire up the replicated cells to the original circuit
5. Construct a simplified graph representation from the updated netlist (including with the added replicas)
6. Insert voters using Johnson's algorithm [8], [9], using the graph we generated in prior step

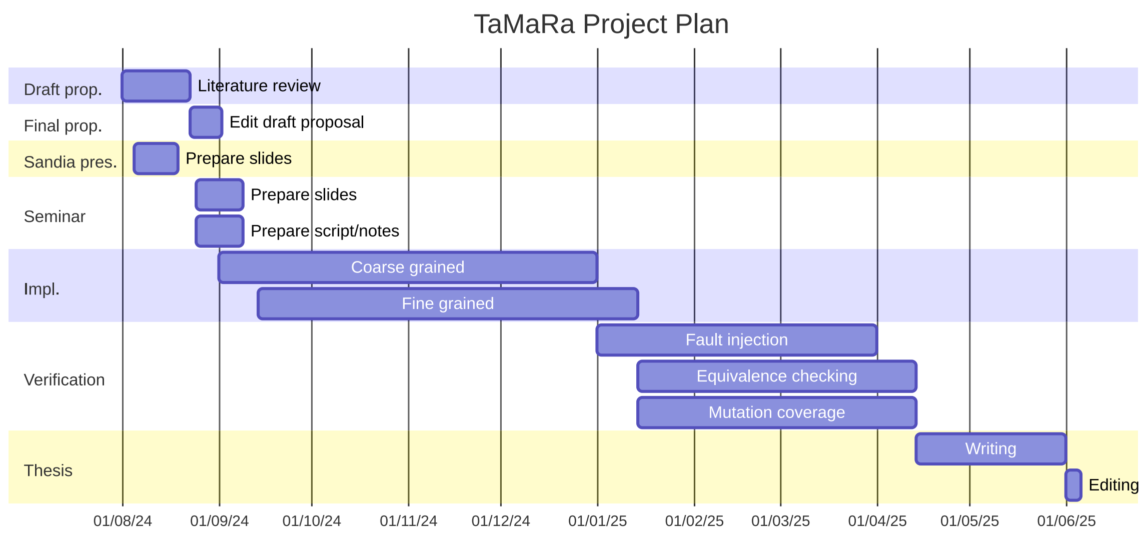
TaMaRa aims to address a number of the limitations identified by authors in the literature review. Kulis [15] identified two important limitations: that TMR may attempt to replicate FPGA primitives that do not have enough physical resources (e.g. trying to replicate a PLL of which only one exists on the FPGA), and that synthesis optimisation may remove the redundant TMR logic. TaMaRa will not attempt to legalise the design, instead, if an FPGA primitive is attempted to be replicated and there are not enough physical resources to back it up, this will be picked up by nextpnr at the placement stage and an error reported. Unfortunately, it may not be possible (or at least, not easily possible) to mitigate the problem entirely, so reporting an error is the best we can do. Since TaMaRa operates after technology mapping, by which time all optimisation has completed, there is no need to worry about the redundant logic being eliminated. This is the key reason for the selection of a netlist-level algorithm, rather than a design-level one.

Some existing approaches [8], [10], [18], [20] do not have a rigorous verification methodology, or a methodology which lacks formal verification. TaMaRa aims to be production grade, so I consider verification to be an important step in the process of ensuring reliability. As mentioned in the engineering requirements and aims, I propose a rigorous verification methodology based on fault-injection simulation *and* formal methods, partially inspired by Benites [11]. Formal verification is widely utilised in industry designs, and is being increasingly researched in academia, so makes sense to use for TaMaRa. In addition to formal verification, I aim to undertake a large-scale fuzzing exercise to further strengthen the quality of verification. Fuzzing in this case will involve the generation and verification of random Verilog RTL in large quantities, rather than just verifying on a few selected testbenches. The verification can be broken down as follows:

- **Fault injection simulation:** Using an open-source simulator like Verilator, Icarus Verilog or cxxrtl, a complex design will have TMR applied to it and be subject to a simulation with controlled random injected SEUs each cycle. I intend to use a simple RISC-V core, either Hazard3 or picorv32 as the Device Under Test (DUT). I plan to use the RISC-V port of CoreMark [25] as a benchmark program and then check the program passes correctly even with SEUs. CoreMark is self-checking, so should fail if an SEU disrupts the processor. It will also have the added bonus of indicating how CPU performance is affected by TMR.
- **Equivalence checking (fuzzing):** This aims to prove that TaMaRa does not change the underlying behaviour of the circuit after it's run. TMR is only supposed to make the circuit redundant, not change its behaviour or timing. The methodology is as follows:
  1. Randomly generate Verilog using Verismith [26]
  2. Run TaMaRa and synthesise the circuit
  3. Use *SymbiYosys* and *eqy* to check that the circuits before and after TMR are identical
- **Mutation coverage:** This aims to prove that TMR works as intended. Ideally, I would also like to use fuzzing to generate random Verilog RTL as in equivalence checking, but the generation of random Verilog with valid testbenches is a topic still under active research in academia. Instead, using carefully selected designs and testbenches, I will use formal verification to prove that TaMaRa removes injected faults. The methodology is as follows:
  1. Pick a simple test case, for example, a parity checker or CRC8 calculator
  2. Write a self checking testbench for the example
  3. Use Yosys *mcy* (mutation coverage) to check that the testbench is valid and high quality
  4. Use TaMaRa to add TMR and synthesise the circuit
  5. Inject faults into the redundant netlist
  6. Use mcy to check that the injected faults are removed by the TMR process

3.4. Milestones and timeline

To design the timeline of the TaMaRa project, I use a Gantt chart, shown below.



3.5. Risk assessment

Before it was known that UQ does not have the facilities to expose an FPGA to real-life radiation, it was considered a possibility that TaMaRa would be tested on a real-life device under intense radiation conditions. This would have created a number of risks and challenges. However, now that this verification approach has been discarded, TaMaRa is a pure software/gateware project, and thus carries no significant health and safety risks.

Nonetheless, TaMaRa is not completely risk-free. Due to the fact that it may be deployed on safety-critical systems, its correct functioning is important. Hence, a simple risk assessment has been prepared.

Risk	**Potential damage**	**Rating**	**Mitigation strategy**
TaMaRa implementation is not able to be completed in time	Thesis result is much worse. Unable to verify results.	Medium	Proper project planning including formulation of engineering requirements and research questions. Regular meetings with supervisor. Contact with YosysHQ dev team.
TaMaRa verification is not able to be completed successfully	Thesis result is worse, not able to prove the TMR algorithm works.	Medium	Research into formal verification and basing work on prior papers. Contact with YosysHQ dev team.
TaMaRa introduces subtle differences in behaviour in the output circuit	Safety-critical systems that TaMaRa is used to design may have unexpected behaviour, potentially leading to severe loss of life or property.	High	Rigorous verification including formal verification and fault-injection simulation.
TaMaRa does not implement TMR correctly	Safety-critical systems that TaMaRa is used to design may fail due to SEUs, causing severe loss of life or property.	High	Rigorous verification including formal verification and fault-injection simulation.

3.6. Ethics

TaMaRa may be used to design defence systems. This is not considered a significant ethical issue.

4. 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. [Online]. Available: http://yosyshq.net/yosys/files/yosys-austrochip2013.pdf

[3] D. Shah, E. Hung, C. Wolf, S. Bazanski, D. Gisselquist, and M. Milanovic, “Yosys+nextpnr: an Open Source Framework from Verilog to Bitstream for Commercial FPGAs,” *CoRR*, 2019, [Online]. Available: http://arxiv.org/abs/1903.10407

[4] 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.

[5] 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.

[6] 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/

[7] K. Zhao, Z. Liu, and F. Yu, “The avenue of FDSOI radiation hardening,” in *2014 12th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, 2014, pp. 1–3. doi: 10.1109/ICSICT.2014.7021436.

[8] 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.

[9] J. Johnson, “Synchronization Voter Insertion Algorithms for FPGA Designs Using Triple Modular Redundancy,” 2010. [Online]. Available: https://scholarsarchive.byu.edu/etd/2068/

[10] 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.

[11] 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.

[12] L. A. C. Benites, “Automated Design Flow for Applying Triple Modular Redundancy in Complex Semi-Custom Digital Integrated Circuits,” 2018.

[13] N. D. Hindman, L. T. Clark, D. W. Patterson, and K. E. Holbert, “Fully Automated, Testable Design of Fine-Grained Triple Mode Redundant Logic,” *IEEE Transactions on Nuclear Science*, vol. 58, no. 6, pp. 3046–3052, Dec. 2011, doi: 10.1109/tns.2011.2169280.

[14] “SkyWater Open Source PDK.” Accessed: Aug. 11, 2024. [Online]. Available: https://github.com/google/skywater-pdk

[15] 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.

[16] N. Engelhardt, “Personal communication.” May 2024.

[17] "Synlig - SystemVerilog support for Yosys." Accessed: Aug. 04, 2024. [Online]. Available: https://github.com/chipsalliance/synlig

[18] 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.

[19] S. Lahti, P. Sjövall, J. Vanne, and T. D. Hämäläinen, "Are We There Yet? A Study on the State of High-Level Synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 5, pp. 898–911, 2019, doi: 10.1109/TCAD.2018.2834439.

[20] A. R. Khatri, A. Hayek, and J. Borcsok, "RASP-TMR: An Automatic and Fast Synthesizable Verilog Code Generator Tool for the Implementation and Evaluation of TMR Approach," *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 8, 2018, doi: 10.14569/IJACSA.2018.090875.

[21] 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.

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

[23] G. Swift and S. Guertin, "In-flight observations of multiple-bit upset in DRAMs," *IEEE Transactions on Nuclear Science*, vol. 47, no. 6, pp. 2386–2391, 2000, doi: 10.1109/23.903781.

[24] A. F. dos Santos, L. A. Tambara, F. Benevenuti, J. Tonfat, and F. L. Kastensmidt, "Applying TMR in Hardware Accelerators Generated by High-Level Synthesis Design Flow for Mitigating Multiple Bit Upsets in SRAM-Based FPGAs," in *Applied Reconfigurable Computing*, S. Wong, A. C. Beck, K. Bertels, and L. Carro, Eds., Cham: Springer International Publishing, 2017, pp. 202–213.

[25] S. Gal-On and M. Levy, "Exploring CoreMark™ – A Benchmark Maximizing Simplicity and Efficacy," 2012. [Online]. Available: https://www.eembc.org/techlit/articles/coremark-whitepaper.pdf

[26] 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.