Fault Tolerant Computing and VLSI Testing

Individual Project:
Self-Purging Redundancy on a MIPS ALU Unit

1 Introduction

Fault tolerant circuit design has become increasingly important over the past several decades, due to the demand for faster and smaller systems. A common method for making circuits tolerate one of the modules developing a fault is through redundancy: having multiple instances of the same module and taking the vote as the majority verdict. These systems would work until the majority of modules developed faults. A system with N modules is an N Module Redundancy (NMR) system.

Following on from NMR, research then investigate hybrid redundancy techniques. These techniques often involve more complex logic to ensure the system remains running even after more than the majority of active modules develop faults. Examples of this include NMR-with-Spare schemes, and Self-Purging Redundancy [1]. The latter scheme will be the focus of this report.

I investigate Self-Purging Redundancy in practice by implementing it on the ALU component of a MIPS processor. I show how the reundancy logic was implemented for our n-bit ALU and in Verilog. Mathematical models for reliability are used to measure how the reliability changes depending on the number of modules and the voter threashold. The circuit was synthesised by the Synopsys Design Compiler and metrics for area, timing and power were taken for different numbers of modules and voter threasholds. From these results, I reach a compromise between reliability and overhead and compare with the original circuit and NMR circuits with the same number of modules. The source code for the final self-purging redundancy scheme is included as part of this submission.

Also included in this report is the scripts used to generate the metrics for our circuits.

2 Realisation of Self-Purging Redundancy

NMR circuits decide on their output using a voter circuit, which use multiple-input AND gates to check if one value is produced by the majority of modules. This is realised by testing all $\binom{n}{k}$ combinations of modules where n is the number of modules and $k = \lceil \frac{n}{2} \rceil$ is the number of modules required for a majority.

We can see this with the example below for three modules a, b, c and output r:

$$r = ab + ac + bc$$

And again for five modules a, b, c, d, e:

$$r = abc + abd + abe + acd + ace + ade + bcd + bce + bde + cde$$

With both circuits, in order for $r = 1$ at least three of the modules must also output 1.

Self-Purging Redundancy also uses a voter, but a more generalised one known as a threashold voter. A threashold voter produces $r = 1$ if at least k modules produce 1, where $k \in \{2, ..., n-1\}$. The version with three modules is the same as before, but for five modules we can now use the following logic for a threashold of two:

$$r = ab + ac + ad + ae + bc + bd + be + cd + ce + de$$

This means that we no longer need a majority of modules to agree on a result.

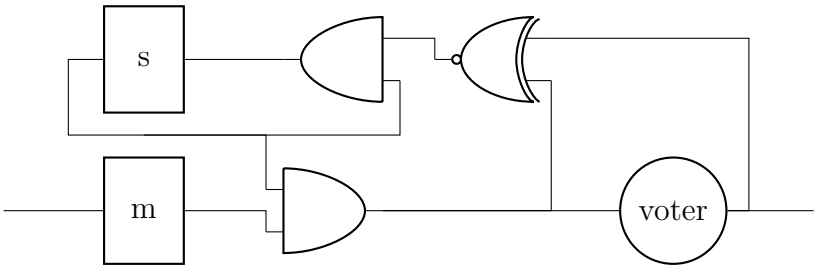
This would not work in traditional NMR, as if more than two modules developed faults then the fault could propagate. Self-Purging Redundancy avoids this by how it handles faulty voters which disagree with the verdict. Instead of allowing them to vote, a 'switch' is used to set that module's vote to 0, preventing it from changing the outcome.

This is typically achieved using an SR flip-flop driven by the system clock, as illustrated by Losq [1, Section 2-B]. Realising this in Verilog however is difficult. Firstly, because the ALU circuit receives no clock input and is instead driven by the pins themselves. And secondly, Verilog's registers are D flip-flops, meaning that we cannot simply drive a reset signal.

For realisation the following compromises were therefore made:

- To avoid the changes required to the overall processor in order to add a clock to the ALU, the flip-flops were driven by the output port.
- To use D flip-flops, an XOR gate was used to set the flip-flop to 0, an extra AND gate was used to ensure that once the switch was set to 0 it could not be set back to 1.

An example of these modifications for a simple module m and D flip-flop s can be seen below. For the sake of simplicity, the driving pin for s has been removed. At initialisation, s is set to 1.



Another interesting question raised from this work was how to move from Self-Purging Redundancy for modules with single-bit outputs such as the one above to modules with multiple outputs, such as our ALU unit which has a 32-bit output pin for the result and a single-bit output pin for zero. The solution used for this project was to treat each output bit independently with its own voting circuit and switch, though another solution could be to switch off the entire module if any of its outputs fail. This was not attempted due to complexity.

Using the above design for the switch and the logic for the voter specified on the previous page, the fault-tolerant ALU was designed in Verilog and synthesised using Synopsys Design Compiler. The design was generalised and tested for three to eleven modules, with voter threasholds ranging from two to $n - 1$ for n modules.

3 Design Compiler Script

The following script was used in Design Vision to synthesise the ALU circuit and measure its performance.

```
#design_vision->File->Read mipsparts.v
current_design alu
compile
report_timing
report_area
report_power
```

4 Results

4.1 Reliability of Self-Purging Redundancy

For measuring the reliability of the self-purging redundancy scheme, the following equation was used from Loch [1, Section 3-A].

$$R_{N,K}(T) = \sum_{i=K}^{N} \binom{N}{i} [R(t)]^i [1 - R(t)]^{N-i}$$

N.B. this formula assumes perfect voter and switch coverage. There is another formula which takes into account faults in the voter [1, Section 3-B], but this one was chosen for simplicity.

References

[1] J. Losq. "A Highly Efficient Redundancy Scheme: Self-Purging Redundancy". In: *Computers, IEEE Transactions on* C-25.6 (June 1976), pp. 569–578. ISSN: 0018-9340. DOI: 10.1109/TC.1976.1674656.