Politecnico di Torino

Testing and Fault Tolerance

Hardening a RI5CY-based Core:
Final Project Report

Master degree in Electronics Engineering

Master degree in Computer Engineering

Referents: Prof. Matteo Sonza Reorda, Riccardo Cantoro

Authors: group 6

Seyed Emadodin Mousavi, s309982
Fabio Gianino, s301556
Ardalan Sohrabian, s314498

January 28, 2024

Contents

CHAPTER 1

Project Specification and the Preliminary Steps

1.1 Introduction to the Assignment 2

In this assignment, we aim to enhance the fault coverage of a digital circuit using hardening techniques solely, without using other methods. We start by augmenting the fault coverage using assembly code, which allows us to utilize the entire circuit as a functional test to identify potential faults in each unit. By employing assembly code, we achieved a 62% increase in fault coverage from the initial 29%. Next, we explore duplication and triple modular redundancy (TMR) approaches, involving replication or triplication of certain module outputs. By comparing the outputs of the replicated modules and identifying discrepancies, we can effectively detect faults. We prefer duplication over triplication for our purpose, as it often suffices for fault detection. is shown in figure **??**.

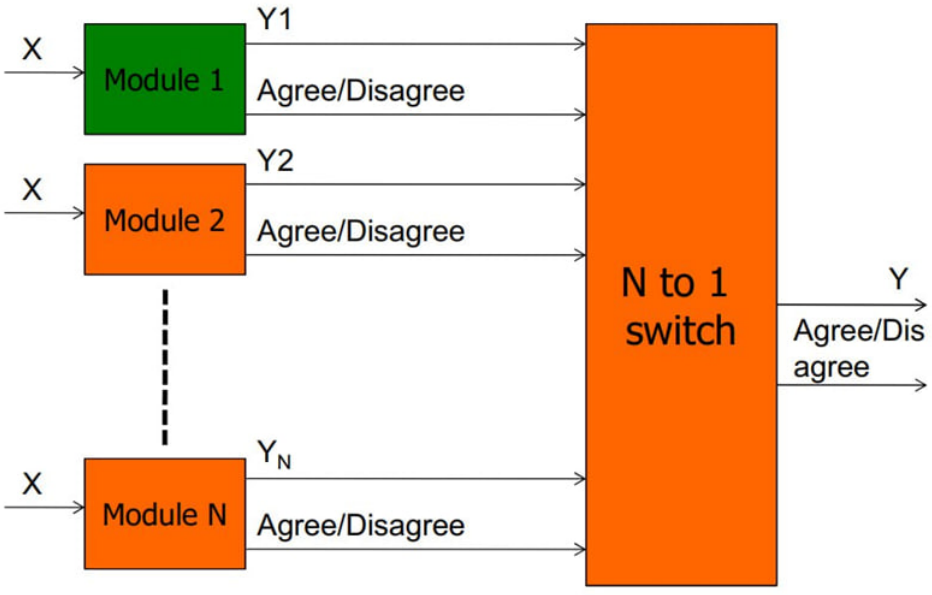

Figure 1.1: Hot Standby Sparing

```
# Cumulative Test Results
# Total Faults:                138672                    138672
# Detected Faults:                 0      0.00%             0      0.00%
# Dropped Detected Faults:       40234    29.01%          40234    29.01%
# Dropped Potential Faults:        684     0.49%            684     0.49%
# Potential Faults:                  0     0.00%              0     0.00%
# Not Detected Faults:           12549     9.05%          12549     9.05%
# Not Strobed:                    1474     1.06%           1474     1.06%
# Not Observed:                  68949    49.72%          68949    49.72%
# Not Controlled:                14782    10.66%          14782    10.66%
#
# Test Coverage:                           29.26%                   29.26%
#
>
```

Figure 1.2: FC of the default project

```
# Cumulative Test Results
# Total Faults:                138672                    138672
# Detected Faults:                 0      0.00%             0      0.00%
# Dropped Detected Faults:       83327    60.09%          83327    60.09%
# Dropped Potential Faults:       1337     0.96%           1337     0.96%
# Potential Faults:                  0     0.00%              0     0.00%
# Not Detected Faults:           13763     9.92%          13763     9.92%
# Not Strobed:                    1474     1.06%           1474     1.06%
# Not Observed:                  31270    22.55%          31270    22.55%
# Not Controlled:                 7501     5.41%           7501     5.41%
#
# Test Coverage:                           60.57%                   60.57%
#
>
```

Figure 1.3: FC of the modified SBST prpgram with the default netlist

1.2 Development of the SBST Program

Next, we implement parity checking by computing parity bits and comparing them to corresponding ones. This enables us to detect faults if any appear.

Finally, we use Hot Standby Sparing (HSS) that are designed to autonomously verify the correctness of operations Hot Standby Sparing is like having backup components ready to take over in case of a failure. This approach enhances the reliability and availability of a system, making it more robust and resilient to hardware faults.

Overall, our hardening techniques have significantly enhanced fault coverage by approximately 20%.

In this phase, our goal is to enhance the resemblance percentage by modifying the SBST (Scan-Based Built-In Self-Test) technique. The aim is to elevate the overall fault coverage from its initial state of 29.26% to an impressive 60.57%. This improvement involves strategic adjustments to the SBST method, allowing us to detect more extensive range of faults within the UUT.

CHAPTER 2

Design and Implementation of the Hardened Core

2.1 The Modified Stages of the Processor

In an effort to make the entirety of the less prone to errors, all the 4 stages of the pipeline of the processor is modified to contain multiple instances of 8X1 HSS for different modules, for example ALU or the Multiplier unit.

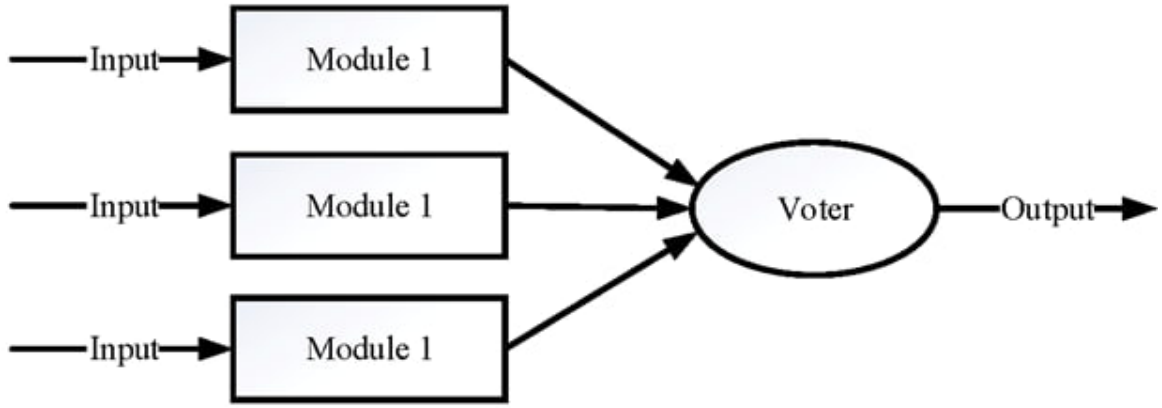

Figure 2.1: The Voting Mechanism

2.2 Final Fault Coverage and other Results

Finally, to make the results more presentable and to achieve the maximum coverage, a combination of scan insertion techniques, along with STIL file based fault simulation was used.

```
    Uncollapsed Stuck Fault Summary Report
-----------------------------------------------
fault class                     code   #faults
------------------------------  ----  ---------
Detected                         DT     460725
  detected_by_simulation         DS    (435156)
  detected_by_implication        DI     (25569)
Possibly detected                PT          0
Undetectable                     UD       2477
  undetectable-unused            UU         (58)
  undetectable-tied              UT        (508)
  undetectable-blocked           UB         (29)
  undetectable-redundant         UR      (1882)
ATPG untestable                  AU        120
  atpg_untestable-not_detected   AN        (120)
Not detected                     ND       2952
  not-controlled                 NC        (678)
  not-observed                   NO      (2274)
-----------------------------------------------
total faults                              466274
test coverage                              99.34%
fault coverage                             98.81%
-----------------------------------------------
          Pattern Summary Report
-----------------------------------------------
#internal patterns                          4731
    #basic_scan patterns                    4731
-----------------------------------------------
(END)
```

Figure 2.2: The Final FC

CHAPTER 3

Comparison of the Results of further Discussions

3.1 Comparison of Basic and Hardened Versions

The basic version of the processor (Figure 3.4) exhibits 138,672 faults, and the initial coverage is approximately 60.55 (3.1). A relatively simple sbst program was utilized for both the basic and hardened versions to facilitate a comparison and evaluate the differences. The final count of potential faults across the entire core is approximately 454,814, with a coverage of about 78.61 (3.2). Although this represents a substantial improvement, it underscores that many hardening solutions significantly increase the circuit area and, consequently, the total number of potential faults. It's crucial to note that these solutions not only enable fault detection but also allow for correction.

Another consideration is that, in the original version, the clock timing is maintained, whereas in the hardened version, the optimized synthesis reveals a maximum negative slack of -0.49. This implies that, to make this new version of the processor functional, a larger clock period will be necessary. The final point to note is the disparity in terms of area, as indicated in the synthesis:

- Initial area: 30,161
- Final area: 94,072

# Cumulative Test Results				
# Total Faults:	138672		138672	
# Detected Faults:	0	0.00%	0	0.00%
# Dropped Detected Faults:	83305	60.07%	83305	60.07%
# Dropped Potential Faults:	1339	0.97%	1339	0.97%
# Potential Faults:	0	0.00%	0	0.00%
# Not Detected Faults:	13742	9.91%	13742	9.91%
# Not Strobed:	1474	1.06%	1474	1.06%
# Not Observed:	31296	22.57%	31296	22.57%
# Not Controlled:	7516	5.42%	7516	5.42%
#				
# Test Coverage:		60.56%		60.56%

Figure 3.1: result of the fault simulation for the initial version of the processor

```
# Cumulative Test Results
# Total Faults:                454814                    454814
# Detected Faults:                  0      0.00%              0      0.00%
# Dropped Detected Faults:     356135     78.30%         356135     78.30%
# Dropped Potential Faults:      2788      0.61%           2788      0.61%
# Potential Faults:                 0      0.00%              0      0.00%
# Not Detected Faults:          28872      6.35%          28872      6.35%
# Not Strobed:                   1880      0.41%           1880      0.41%
# Not Observed:                 48045     10.56%          48045     10.56%
# Not Controlled:               17094      3.76%          17094      3.76%
#
# Test Coverage:                           78.61%                    78.61%
```

Figure 3.2: fault simulation on the last version of the processor

Figure 3.3: Comparison between the two fault simulations

3.2 Discussion and Observation

The main result by experiencing the modification of the processor has been to find that the real impact on fault coverage is achieved through components that contain the majority of those faults, as expected. These main actors are the Register File, the ALU, and the Multiplier. By just applying the simplest possible solution, the parity bit for the RF and duplication with comparison for the others, the increase of coverage was already quite large. A 4% for the parity on RF, and 6% on ALU duplication. By hardening the processor, we can actually increase the fault coverage, but it is always a tradeoff choice where multiple variables must be considered, depending on the final objective. Solutions with hardening impact most of the time the size of the circuitry, resulting in more and more power consumption, lowering of the performance, and obviously the final cost of the system. Moreover, the number of strobe points will impact the final cost because the resulting testing phase with an ATE increases in cost along with that number. But those solutions are also really impactive to achieve the maximum possible fault coverage.

Nonetheless, every solution has a boundary where introducing additional strobe points, beyond a specific juncture, fails to yield substantial improvements. The impact becomes lower and lower.

In the end, to make things better, we decided to use another hardening solution, even if not belonging to the Fault Tolerance one: scan chain. It allowed us to cover more than we could before.

3.3 Possible Future Improvements

In our project, we worked on many parts inside the core, but there's room for more improvements. For instance, Hamming codes can find and fix multiple errors in the Register File, but they take up more space on the circuit. Using a parity bit can probably be the best solution in terms of resource costs, but it is limited to single errors and does not provide correction. Not only Hamming codes but also checksum solutions can be applied to every synchronous module or signal that works as registers. The second way is to duplicate or use similar approaches in all the combinational blocks, and obviously, the more the strobe points, the more the coverage increases. But choosing this last method won't really make things better, considering the bigger circuit, performance problems, and so on.

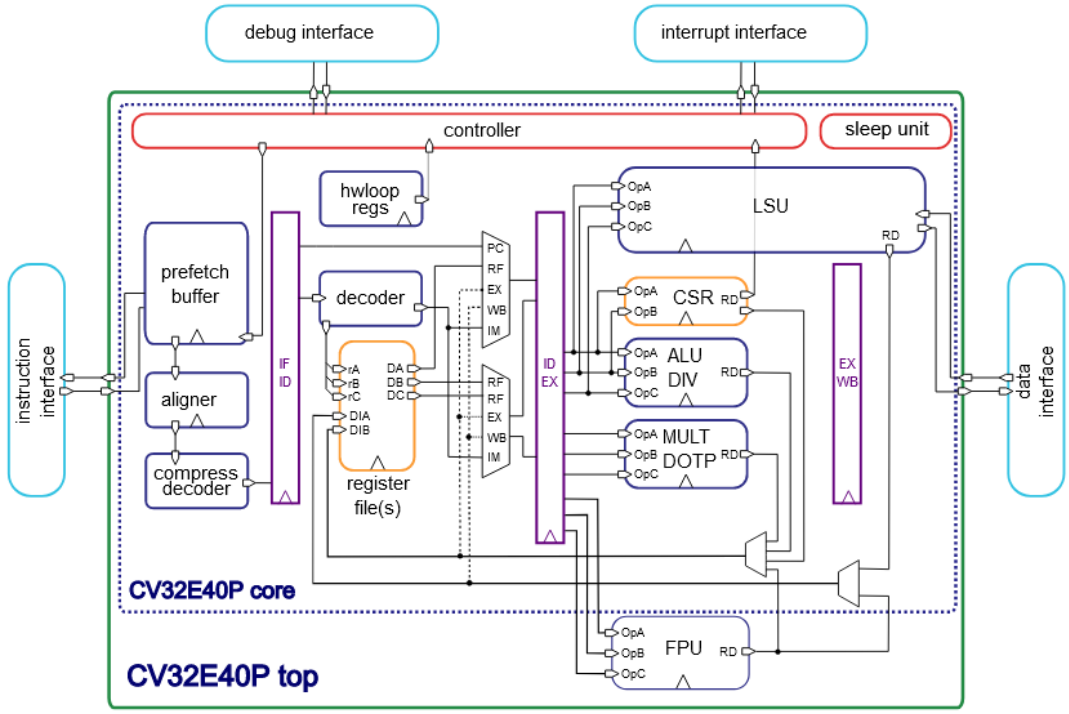

Figure 3.4: Schematic of the processor