P1: Systematic Analysis for System-Level Test Fails

Milestone Report – GS-IMTR

Nourhan Elhamawy

October 18, 2022

Advisor / main examiner: Prof. Dr. Jens Anders & Prof. Dr. rer. nat. habil. Ilia Polian

Co-examiner: Prof. Dr. rer. nat. Dirk Pflüger

Mentor: Dr. Matthias Sauer

Acronyms

ATPG Automatic Test Pattern Generator/Generation 3

eVCD Extended Value Change Dump 5

FT Final Test 7

GPDK General Process Design Kit 4

ISA Instruction Set Architecture 4

SLT System-Level Test 2

STIL Standard Tester Interface Language 5

1 Introduction

With the increasing need for complex systems and the increasing reliance on electronic devices in critical applications e.g aviation, medical devices etc., the demand for quality of the systems is as high as ever. And since testing is the only way to achieve such high quality. the testing flow is being continuously adapted and refined. As indicated in Figure 1, there are a number of tests performed to ensure the quality of the manufactured devices. System-Level Test (SLT) is the outgoing quality measure performed before a device is deployed in its operation environment as described in [1]. Therefore, the system is tested as one entity, where the software is running on the hardware to mimic its normal operation mode.

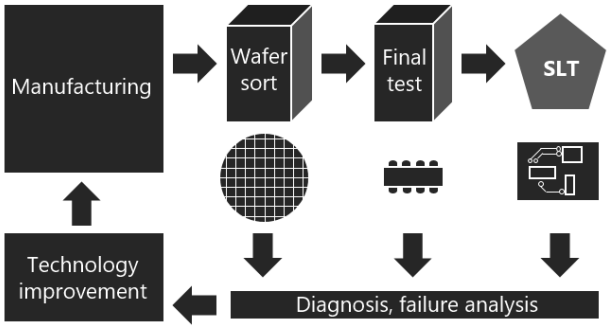

Figure 1: Standard test flow

SLT has been added as a last test insertion within the standard step when required test coverage was not reached with the structural and functional tests [2]. In Figure 2, the venn diagram shows that maximum test coverage can only be reached when structural, functional and SLT are performed.

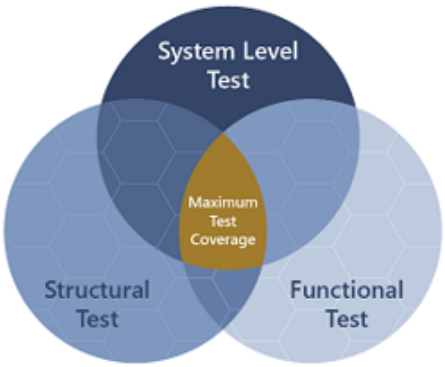

Figure 2: Venn diagram for maximizing test coverage

As mentioned before, SLT mimics a system in its normal operation mode and views it as one entity, as opposed to the previous testing steps, where each building block of a system has been tested on its own. We aim to close this gap between the different building blocks. By appling and refining the methods of SLT, we want to demystify the mysteries associated with SLT. Some ideas as to why marginal defects manifest and can only be caught with the help of SLT have been developed but still not concluded. Our target is to explore the reasons and find possible solutions with the help of SLT to catch and mitigate marginal defects.

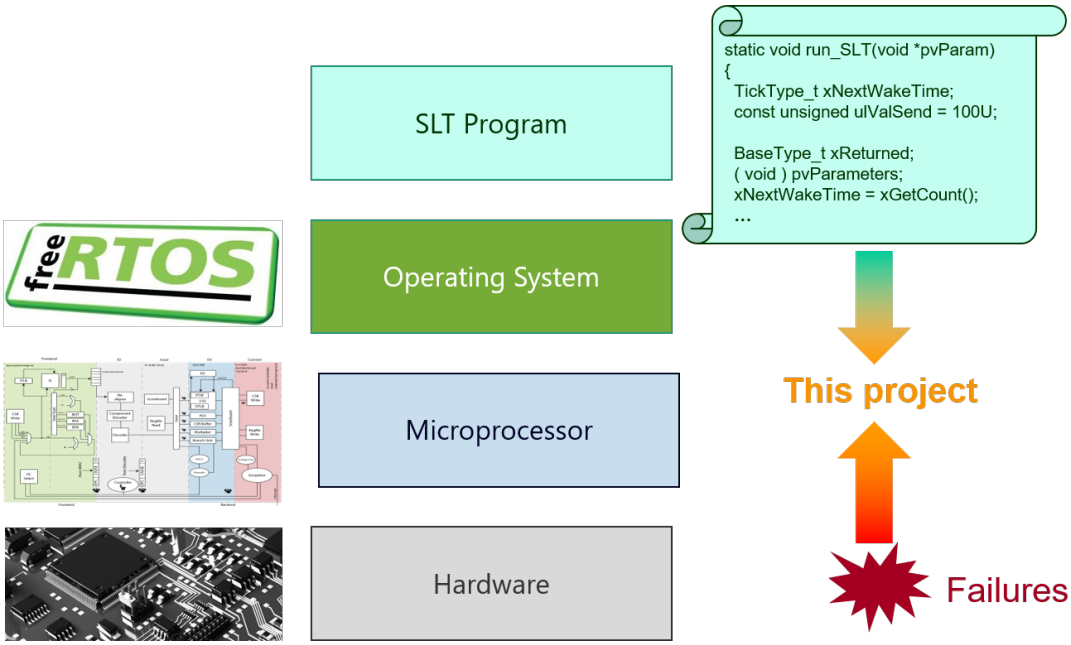

Figure 3: This project

2 Implementation

The purpose of this framework is to have a testing platform in order to investigate causes of marginal defects. Moreover, the framework is implemented using commercial tools to make it easy to adopt into a running system. It also has two modes of operation: the Automatic Test Pattern Generator/Generation (ATPG) mode and the SLT mode. Figure 4 shows the complete framework. A list of the utilized tools is found under Subsection 2.3

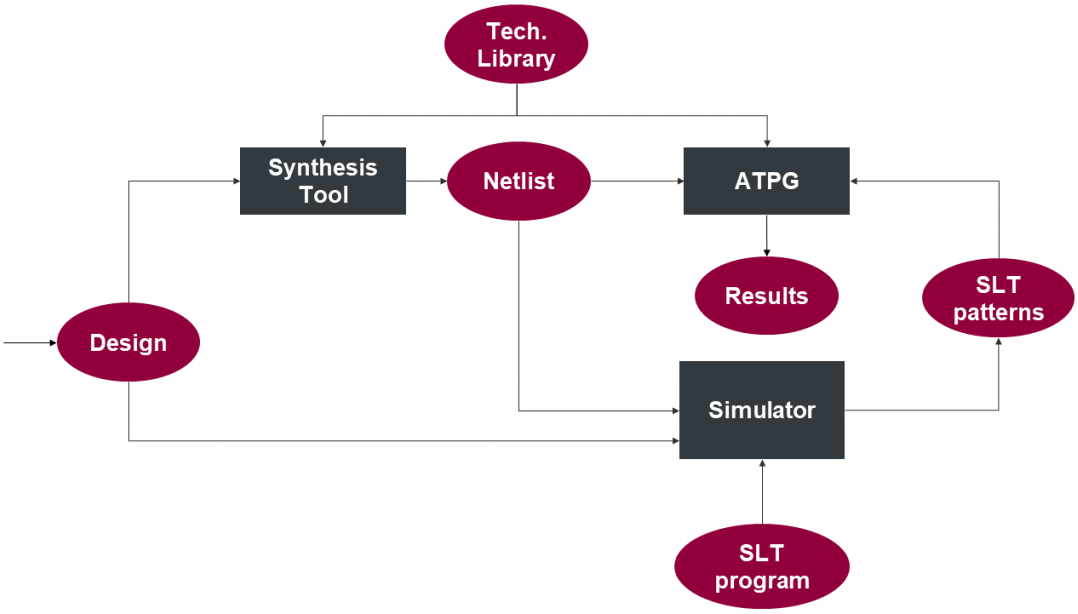

Figure 4: The current framework.

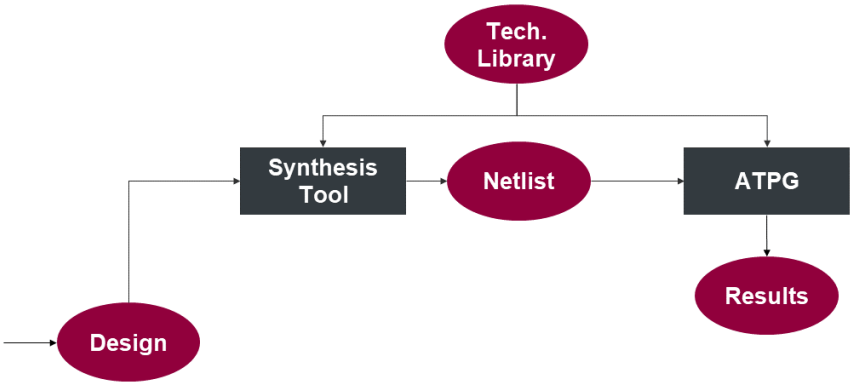

Figure 5: The ATPG framework.

2.1 The ATPG framework

As mentioned already, the framework has two modes: the ATPG mode shown in Figure 5. The first step is to synthesize the design to a gate-level netlist with the help of a technology library. Then the ATPG can take in the gate-level netlist and the technology library in order to perform the structural test and outputs the results. The results can be the fault coverage, the test coverage, the fault list as well as the patterns that have been applied.

2.2 The SLT framework

The SLT mode of the framework extends the ATPG mode to give the complete framework as shown in Figure 4. It is extended to include a logic Simulator that simulates the SLT program and outputs the SLT patterns. The ATPG accepts the SLT patterns together with the gate-level netlist and the technology library to perform a fault simulation. The results are also the same as in the ATPG mode.

2.3 Tools

The idea of having this framework is to have a testing platform for different designs and different fault models. Therefore, we are using commercial tools to build the SLT framework.

- Design: The CVA6 Core [3] based on the RISC-V Instruction Set Architecture (ISA)
- Synthesis Tool: Design Compiler by Synopsys
- Technology library: Cadence General Process Design Kit (GPDK)45 nm
- Logic Simulator: QuestaSim by Siemenes Mentor Graphics
- ATPG/ Fault Simulator: TestMax by Synopsys

2.4 Challenges & Solutions

While implementing the framework, a lot of challenges have been encountered. In this section, some of these challenges are listed with some of the applied solutions to circumvent these obstacles.

1. Find a RISC-V core that meets the requirements of being able to boot an operating system, easily extended to multi-core and runs in our simulation
   - Multiple candidates UC Berkley's Rocket, SiFive's U54, SiFive's E31 and the ETH Zurich's CVA6. The CVA6 simulation ran with little effort.
2. Genus Synthesis Tool optimized the core by removing pipeline stages
   - To solve this problem we simply switched to another Synthesis Tool, Design Compiler, that gave a better suited performance to our needs.
3. Logic simulation time

- Use memory pre-loading instead of the debugging module

4. Fault simulation time & low fault coverage
   - Use memory template for modelling the memory in ATPG
   - Scan-chain insertion
5. Extended Value Change Dump (eVCD) file dumped by QuestaSim gives wrong values
   - Extract the values as List file
6. eVCD file does not include complete scan pattern information
   - Translate to Standard Tester Interface Language (STIL) format to match a scan pattern as indicated in Figure 6

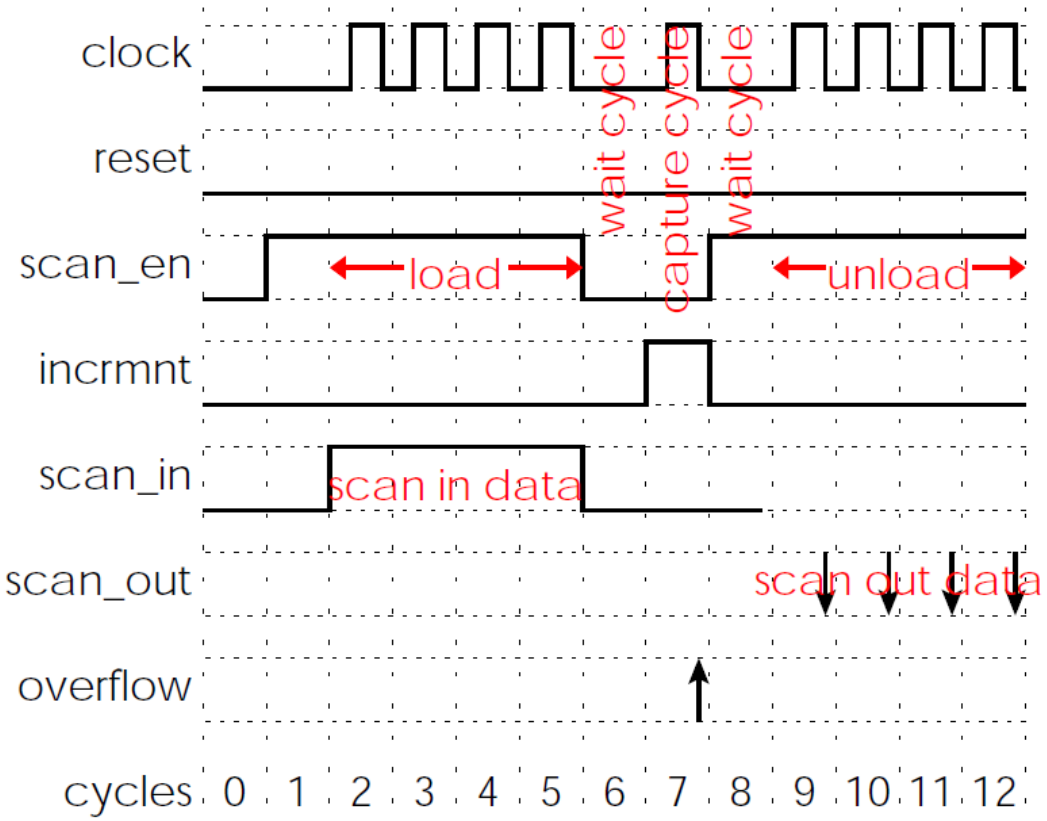

Figure 6: Example of a scan pattern

3 Results

3.1 ATPG Results

Module	Without scan-chains			With scan-chains		
	Test cov.[%]	Fault cov.[%]	# Faults	Test cov.[%]	Fault cov.[%]	# Faults
ariane	0.17	0.17	1,409,028	98.75	97.99	1,465,876
⌊_ commit_stage	100	87.51	6,792	100	87.43	6,792
⌊_ controller	100	28.12	384	100	28.06	392
⌊_ csr_regfile	96.75	94.06	71,490	100	95.59	81,292
⌊_ ex_stage	64.74	63.63	637,926	99.90	98.22	649,386
⌊_ frontend	84.71	84.20	119,258	100	99.40	135,882
⌊_ id_stage	99.95	97.47	15,230	99.97	97.59	16,058
⌊_ issue_stage	95.61	95.46	405,776	100	99.92	427,906
⌊_ perf_counters	100	95.32	41,262	100	91.59	54,546
⌊_ wt_cache_subsystem	12.02	11.91	168,402	>90	crashed	173,720

Table 1: ATPG resutls: with and without scan-chains

The results of the ATPG Simulation are shown in Table 1. Here the fault and test coverages are listed for the design first without scan-chains [4] and with scan-chains. The results for the design including scan-chains show an increase in fault and test coverage for all sub-modules of the processor as well as a slight increase in the number of faults which are introduced because of the extra hardware for the scan-chains. For the processor without scan-chains the ATPG Simulation time exceeded 2 weeks without much progress. As for the cache subsystem with scan-chains the simulation failed to terminate correctly but the progress showed a higher than 90 % test coverage for that module.

3.2 SLT Results

Module	Test cov.[%]	Fault cov.[%]
ariane	?	?
⌊_ commit_stage	7.33	6.41
⌊_ controller	93.52	26.30
⌊_ csr_regfile	36.88	35.85
⌊_ ex_stage	?	?
⌊_ frontend	20.83	20.71
⌊_ id_stage	70.42	68.68
⌊_ issue_stage	58.24	58.21
⌊_ perf_counters	11.93	11.37
⌊_ wt_cache_subsystem	?	?

Table 2: SLT Results for the CVA6 without scan-chains

Table 2 shows the results for the SLT patterns when applied to the CVA6 Core without inserting scan-chains. As indicated, the results show low coverages for test and fault coverages in comparison to the ATPG results. For some module simulation did not terminate for over a week without much progress. These modules are listed in rows 1,5 and 10.
The SLT results for the design with scan-chain should be ready as soon as Challenge # 6 is resolved. We expect a much better performance with scan-chains as was the case with the ATPG results.

4 Application

One application that has been developed with this framework is a solution for the "Long-Tail Test Patterns ". A tester's memory is usually limited in size and the Final Test Insertion patterns are often more than what a tester can store in a single insertion. Therefore, two solutions are applied to overcome this problem. Either the tester is run multiple times, each time inserting the next set of patterns. This causes an increase in test time based on the number of insertions required to apply all the test patterns as well as an increase in test cost. Or the test patterns are ordered in descending order according to their number of fault detection and the test patterns with the least number of fault detection are chopped off, resulting in loss of fault coverage. These patterns are called the "Long-Tail Test Patterns" according to how they look as shown in Figure 8.
With the help of this framework we have been able to find a solution that maintains the fault coverage, reduced the test time and is optimized for cost. This is achieved by merging the two modes of the framework to reduce the number of ATPG patterns needed to reach the desired fault coverage. First, the SLT patterns are applied to extract the fault list covered by the SLT program. These faults are then exclu

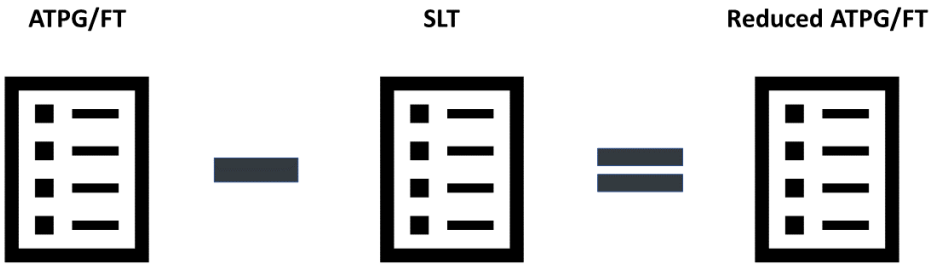

Figure 7: Multi-Layer Flow

Figure 8 illustrates the results of the multi-layer simulation for the controller sub-module. On the x-axis the ATPG patterns as originally numbered are shown. On the y-axis the number of faults are displayed. The red vertical line is a hypothetical limit of a tester's memory set at 75% of the number of patterns. The blue bars represent the ATPG patterns that uniquely detect faults not detected by the SLT patterns. While the pink bars represent the patterns which overlap in fault detection with the SLT patterns. This serves as a proof of concept with a very basic SLT program such as "Hello,World!". According to this example, after the multi-layer simulation only one Final Test (FT) insertion is needed where only two patterns are applied instead of the two original insertions containing in total 13 patterns.

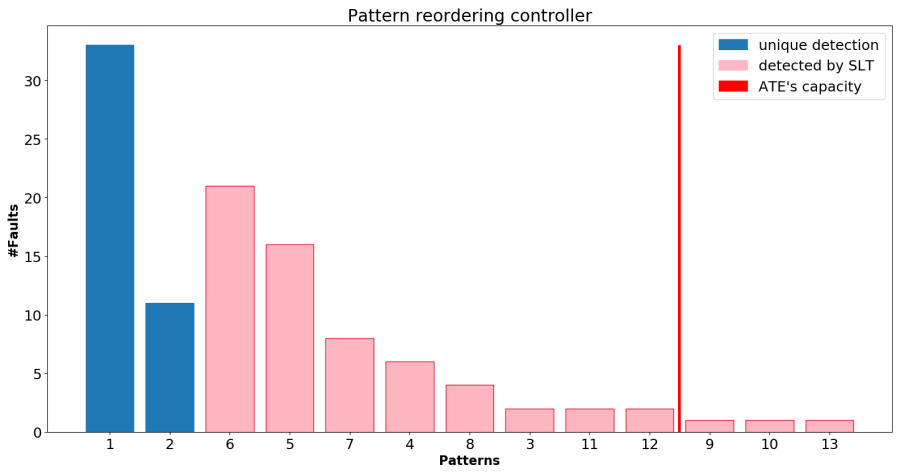

Figure 8: Long-Tail: Controller Module

4.1 Cost estimation

Testing cost is currently the dominating factor when it comes to a device's cost. Therefore, we measure the effectiveness of our method by calculating its cost optimization in comparison to the traditional test methods. The cost function we use to calculate the cost of the multi-layer simulation is described as $c(n,p) = P_{pass_{FT_i}} \times (p \times x_1 + n \times x_2) + P_{pass_{SLT}} \times b \times (y_1 + t \times y_2) + z_1 \times (1 - Y^{1-f_{cov_{FT+SLT}}})$. where

x_1 = 10: FT cost per pattern per device

x_2 = 100: FT cost per insertion per device

y_1 = 1, 000: cost FT to SLT per device

y_2 = 10, 000: SLT cost per test program per device

t : number of test programs (currently 1)

z_1 = 100, 000: cost per test escape

p : FT pattern count

n : number of insertions: $\lceil \frac{p}{max_p} \rceil$

b : boolean variable $\in \{0, 1\}$

A device can only enter a testing phase if it has passed the previous one. Which means we need to include the conditional probability of passing the previous insertion. We differentiate between the probability of passing a test insertion and P_{pass} and P_{fail} for each test insertion whether in FT or in SLT.

5 Conclusion and Next Steps

We have presented our work regarding SLT and the framework being built to facilitate the application of SLT. We showed one application of the framework to reduce the number of patterns applied in FT insertion which in turn reduces the cost of testing. We also included the cost function estimation which we intend to refine in order to generate accurate cost estimations. Moreover, as soon as the last challenge is overcome within the framework, it will offer a tool for further investigating complex design models as well as more elaborated fault models to explore possible reasons for marginal defects.

References

[1] Ilia Polian, Jens Anders, Steffen Becker, Paolo Bernardi, Krishnendu Chakrabarty, Nourhan El-Hamawy, Matthias Sauer, Adit Singh, Matteo Sonza Reorda, and Stefan Wagner. Exploring the mysteries of system-level test. In *2020 IEEE 29th Asian Test Symposium (ATS)*, pages 1–6, 2020.

[2] Harry H. Chen. Beyond structural test, the rising need for system-level test. In *2018 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, pages 1–4, 2018.

[3] F. Zaruba and L. Benini. The cost of application-class processing: Energy and performance analysis of a linux-ready 1.7-ghz 64-bit risc-v core in 22-nm fdsoi technology. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(11):2629–2640, Nov 2019.

[4] Jaume Abella, Sergi Alcaide, Jens Anders, Francisco Bas, Steffen Becker, Elke De Mulder, Nourhan Elhamawy, Frank K. Gürkaynak, Helena Handschuh, Carles Hernandez, Mike Hutter, Leonidas Kosmidis, Ilia Polian, Matthias Sauer, Stefan Wagner, and Francesco Regazzoni. Security, reliability and test aspects of the risc-v ecosystem. In *2021 IEEE European Test Symposium (ETS)*, pages 1–10, 2021.