Fault, an Open Source DFT Toolchain

Mohamed Gaber, Manar Abdelatty, and Mohamed Shalan

The American University in Cairo, EGYPT

Abstract – The continuous reduction in feature size increases the probability that a manufacturing defect in the integrated circuit will result in a faulty chip. Design for Testability (DFT) makes it possible to detect all faults in a circuit after fabrication; and hence reducing the time and cost associated with chip development. However, despite the maturity of the DFT field, there is no practical open source DFT solution. In this paper, we introduce Fault, an open source toolchain for automatic test pattern generation (ATPG), scan insertion and scan chain testing.

keywords: VLSI, EDA, DFT, ATPG, Scan Insertion, Scan Chain, JTAG, Open Source, Fault, Defect, Stuck-at

Introduction

However circuits are designed to be error-free, manufactured circuits may not function correctly because the manufacturing process is not perfect. Defects, such as short circuits and open-circuits, may be introduced. Because of that, testing manufactured circuits became a must-do step. It is crucial to identify faulty circuits as early as possible. Because when the faulty chip is soldered on a printed circuit board, the cost of fault remedy would be multiplied by ten. And this cost factors continues to apply until the system has been assembled and packaged and then sent to users.

Testing of digital logic involves the application of test data (test pattern/vector) to the Device Under-Test (DUT) and the comparison of the resulting response to the expected one. Manufacturing defects tend to alter the circuit behavior; hence, faulty circuits produce incorrect results. Test pattern generation is a complex process with three main aspects: the cost of test generation (generation time), the cost of test application (testing time) and the quality of test (coverage).

The many possible issues that may arise during the fabrication of a hardware design requires almost any hardware written to be designed with testability in mind. To this extent, standards have been introduced to assist with automated testing, most famously IEEE 1149.1 (1). There are a number of tools that are able to generate the infrastructure necessary for that standard, but there is a surprising dearth of open source utilities for such.

The importance of a robust DFT tools in an open source digital design flow cannot be understated. The EDA research team at the American University in Cairo has thus endeavored to create such a tool: one that leverages existing open source tools such as the Yosys Open SYnthesis Suite (2), the Icarus Verilog simulator (3) and Pyverilog (4) to deliver a cohesive experience encompassing synthesis, netlist cutting, test pattern generation, compaction all the way to scan chain stitching and verification. We call this toolchain Fault. Fault is designed and implemented to support standard EDA format; hence, it can be integrated into any industrial RTL to GDSII flow.

Fault Toolchain Design

Fault operates on RTL designs in Verilog and is made up of five components: `synth`, `cut`, `PGen`, `compact`, and `chain`. The Verilog RTL is first synthesized into a flatten netlist using `synth`. Then, the design is converted into pure combinational design using `cut` by cutting out the flip flops and replace them by input and output ports. This modified netlist is used for the ATPG process done by `PGen`. The generated test vector set is then compacted by `compact`. Finally, scan chain insertion is done by `chain`.

Synth is a synthesis script based for Yosys that synthesize and map Verilog RTL design into a flattened netlist that can be used with the subsequent tools of the Fault toolchain. Fault is compatible with any flat netlist, of course, so this step can be skipped if a user would elect to run their own synthesis script.

Cut removes the flip flops from the netlist and converts the design into pure combinational design. The new netlist has an extra input port for every removed flip-flop output pin. Also, it has an extra output port for every removed flip-flop input pin. This step is essential for automatic test pattern generation.

PGen is an automatic test pattern generator (ATPG) for stuck-at faults. Main uses pseudo-random ATPG coupled with fault simulation. This is a simpler alternative to algorithmic methods such as PODEM and D algorithms. Algorithmic methods require "path sensitization" which makes them complex to handle netlists mapped using any arbitrary standard cell library. In PGen, test patterns are pseudo-randomly generated, and a testbench is generated for each test pattern that compares a golden model to a model where fault sites are progressively simulated using Verilog `force` statements. The outputs of both models are compared, and any fault site that can be marked as detectable using said test pattern is sent back to Fault to be marked as covered. Final coverage is then output to a file in a JSON format.

Compact reduces the count of the test vector set using static compaction. Compaction starts with two sets: the initial test vector set, generated by the ATPG, along with its covered faults and an empty compacted test vector set. Firstly, test vectors that cover essential faults, covered by only one vector, are added to the compacted set. Then, the test vector with the highest number of detectable faults is inserted in the compacted set and the faults covered by that vector

Table 1. Benchmark Results.

		LFSR					Swift				
Benchmark	Gate Count	Coverage (%)	Run-time (sec)	TVCount	Compacted	CPercent (%)	Coverage (%)	Run-time (sec)	TVCount	Compacted	CPercent (%)
c6288	2348	99.99	23.51	100	26	74.00	99.97	23.91	100	25	75.00
c5315	1388	98.09	9.8	100	49	51.00	98.40	9.49	100	52	48.00
c5315a	1332	97.75	9.7	100	46	54.00	98.50	9.71	100	46	54.00
c3540	982	95.32	20.46	300	58	80.67	97.63	20.07	300	65	78.33
s15850a	2792	90.32	119.63	400	91	77.25	90.28	121.83	400	88	78.00
s5378a	1030	92.32	35.04	400	71	82.25	94.74	33.84	400	79	80.25
s1488	610	96.60	12.98	300	57	81.00	95.23	8.51	200	52	74.00
s1423a	481	97.74	8.29	200	37	81.50	95.41	8.42	200	34	83.00
s1423	481	95.54	4.45	100	34	66.00	95.48	9.01	200	31	84.50
APU	5121	83.59	175.86	400	114	71.50	86.017	178.32	400	114	71.50
zigzag	4580	99.99	92.32	100	22	78.00	99.99	95.1	100	22	78.00
RGB2YCBCR	3568	91.40	117.33	400	62	84.50	91.89	118.97	400	66	83.50
cpu6502	2395	92.70	71.99	400	113	71.75	91.83	71.58	400	115	71.25
can	2248	91.73	73.74	400	120	70.00	90.75	75.15	400	112	72.00
cic_interpolator	1026	98.18	31.89	400	23	94.25	98.53	23.46	300	23	92.33

are removed from the initial test vector set. This process is repeated until the compacted set covers all detectable faults. The compacted set coverage is then output to a file in a JSON format. This step is important for reducing the cost of testing.

Chain performs scan-chain stitching. Using Pyverilog, a scan-chain is constructed through a netlist's D-flipflops and on the netlist's ports, going input, internal flipflops, then output. Chain offers an option to resynthesize after stitching the scan chain, but again, a user may elect to run their own synthesis on the stitched model. Additionally, chain offers an option to verify its own scan chain to ensure its integrity.

Implementation

Fault is implemented in the Swift Programming Language (5). The reason for this is that a statically-typed, safe, native language that could also interact with and use Python-based libraries was required, and the team leadership is quite experienced in the Swift programming language.

Fault thus interacts with Pyverilog directly via the Swift-Python interoperability developed by Google Inc. as part of their Swift for Tensorflow project (6), and interfaces with Yosys and Icarus Verilog via a shell interface: parsing their outputs via stdin, and stdout. All the glue logic is implemented in pure Swift, which greatly contributes to the swiftness and stability to the tool in comparison to developing the toolchain in pure Python.

Unfortunately, the interoperability itself has been known to introduce the occasional issue as Python struggles against Swift's ownership model: namely, Swift's references counter vs Python's. One example is that using Python in a multi-threaded context from Swift has been known to make the application crash. This is a problem as the simulations are indeed multi-threaded: the Python components were isolated for such a reason.

Additionally, a more practical problem is that setting up the Swift language, let alone Swift/Python interop, is rather cumbersome especially on Linux, where cloud infrastructure typically lies. To alleviate these issues, lightweight Docker images available for instant deployment were created so one may run Fault reliably and without configuration issues.

Benchmarks

Fault's performance was evaluated on a selected number of benchmark, open-source, designs encompassing sequential and combinational element. In order to study the performance of different pseudo random number generators, the test vector set was generated twice using two different random number generators; swift's system random number generator and a 32-bit linear feedback shift register (LFSR).

Table 1 shows the experiment results on the benchmarks including the ISCAS 85 combinational and the ISCAS 89 sequential benchmark circuits. The table shows the gate count, final coverage, run time, the count of both the initially generated test vector set and the compacted set and the compaction percentage. The benchmark designs were synthesized using the osu035 standard cell library. For the designs containing sequential elements, flip-flops were exposed as ports using the cut option. The number of test vectors was increased incrementally to reach a minimum coverage of 95% , but if the number of the generated test vectors hit the specified count ceiling , the simulation settled for the ceiling coverage.

The experiments were carried out on the Precision 5820 workstation using Fault's Docker image on Windows host with an allocated memory of 13568 MB and 12 CPUs.

Conclusion and Future Work

We believe we have developed a rather competent tool for DFT. To our knowledge, Fault is the only open-source DFT solution which can be applied to HDL designs. For future work, we are planning to extend Fault to introduce transition delay faults. Also, Fault will be updated to produce output files understood by commercial ATE. Finally, we are going to evaluate different pseudo-random test pattern generation algorithms to, concurrently, improve fault coverage and reduce the generation time.

Acknowledgment

Fault's development has been assisted by the Cloud V project, a sister project at the American University in Cairo, and its code and Docker container are hosted in Cloud V repositories.

The Fault repository and complete source code are available at https://github.com/Cloud-V/Fault, as well as the benchmarks used for testing. The Benchmarks file contains the results table for all the benchmark designs. Usage instructions are available in the Readme file. It has been tested to work with Apple macOS and GNU Linux (Ubuntu 18.04). Due to the complicated set of dependencies required to run the Fault toolchain, a Docker image is available for use which works universally.

References

1. Ieee standard for test access port and boundary-scan architecture. *IEEE Std 1149.1-2013 (Revision of IEEE Std 1149.1-2001)*, pages 1–444, May 2013. doi: 10.1109/IEEESTD.2013.6515989.
2. Clifford Wolf. Yosys open synthesis suite. http://www.clifford.at/yosys/.
3. Stephen Williams. Icarus verilog. http://iverilog.icarus.com.
4. Shinya Takamaeda-Yamazaki. Pyverilog: A python-based hardware design processing toolkit for verilog hdl. In *Applied Reconfigurable Computing*, volume 9040 of *Lecture Notes in Computer Science*, pages 451–460. Springer International Publishing, Apr 2015. doi: 10.1007/978-3-319-16214-0_42.
5. Chris Lattner and Apple Inc. The swift programming language. https://swift.org/.
6. Google Inc. Swift for tensorflow. https://www.tensorflow.org/swift.