DOI 10.31114/2078-7707-2018-2-99-101

Using Formal Coverage Analyzer for Code Coverage Improvement

Y.A. Tatarnikov

Independent Contractor, yuritatar@gmail.com

Abstract — **This presentation summarizes the results of using the tool grounded on formal proof technics (in this case the tool is Synopsys Formal Coverage Analyzer (FCA)) to improve code coverage for two design Blocks.**

The goal is to find unreachable coverage constructs (UNRs) in the target design Blocks and remove them from the list of uncovered constructs. The removal of UNRs saves the Designers and Verification Engineers the time needed to achieve high level of code coverage.

FCA became part of the design and verification methodology within our organization following its successful evaluation.

Keywords — **Design Verification, Formal Verification, SystemVerilog, SVA.**

I. Introduction

At the end of verification, when you created and debugged all tests from your test plan your block has some level of coverage, but you need to reach ~ 100% code coverage.

Most popular code coverage metrics are: line, toggle, condition, FSM. Ultimately, you have to have 100% for each of them.

It is responsibility of block Designer to get high level code coverage. For it Designer makes the decisions of: adding some tests OR to exclude some not covered constructs. Verification guy helps him/her providing existing coverage data base and coverage reports, implementing new tests scenarios and/or excluding coverage constructs. This process is iterative and takes pretty much time, because usually you start with thousands of uncovered constructs.

Any means to shorten the list of uncovered constructs are welcome. Fortunately, we can get help from Vendors Formal Proof tools, which usually have the special mode to get unreachable coverage constructs (called UNRs), which cannot be covered by any test for this particular block. One of the tools is Synopsys Formal Coverage Analyzer (FCA), which is part of Verification Compiler [1, 2].

Goal of this paper is to show briefly how to use this tool and which results we got, why we included this tool in our Design flow.

II. Use flow for getting UNRs

- You (verification guy) – run regression with coverage enabled to get Coverage Data Base.
- FCA has Coverage Data Base as its input.
- You provide clock(s), reset(s) and reset durability OR give simulation snapshot, which represents your Design initial state.
- FCA selects uncovered coverage constructs (for line, toggle, condition, FSM metrics) and tries to generate timing diagram, which will cover them.
- If it can not find – this construct is uncoverable.
- FCA generates file with all unreachable constructs (UNR).
- You put this file in the command, which reports coverage, to exclude UNRs.

III. When is proper time to do this job

For FCA - less uncovered constructs - better.

For Verification Engineer proper time is when you have finished the creation and debugging tests according to your Test Plan. With good, detailed test plan code coverage metrics (line, toggle, condition, FSM) might be on the level 70%-80%.

IV. Blocks used for FCA evaluation

There are 2 blocks, which were under development at the time of evaluation. Let's call them "block A" and "block B".

In the terms of coverage constructs, block A looks like:

1. **Lines - 1835**
2. **Conditions - 473**
3. **Signal bits (for toggle) – 27974**
4. **FSM states – 61**

As you can see this Block is pretty small, but functionally not easy.

Block B:

1. **Lines - 43988**
2. **Conditions - 13470**
3. **Signal bits (for toggle) – 672904**
4. **FSM states – 72**

Block B looks ~ 20-30 times greater than block A.

V. Blocks A and B coverage before FCA

Table below shows coverage numbers of both blocks. Fractions xxx/yyy in the table mean: xxx – not covered constructs, yyy- total amount of constructs, (zz%) - percentage of uncovered constructs.

Table 1

Code coverage before FCA

	Block A	**Block B**
Lines	**123/1835 (7%)**	**7035/43988 (16%)**
Conditions	**57/473 (12%)**	**2377/13470 (18%)**
Toggle (bits)	**315/27974 (1%)**	**9013/672904 (1.5%)**
FSM states	**1/61 (1.5%)**	**3/72 (4%)**
Total	**496/30343**	**18428/730434**

Comments to the table 1 (above):

- **toggle** is counted for each bit of each block signal. If bit has both transitions: 0->1 and 1->0 - this bit is counted as toggled. If one transition or no transitions – no toggle for this bit.
- **condition** is one particular combination of input signals for given expression. If, for example, we have expression in source code (a && b) and for coverage we have 3 combinations of input signals: 11, 01, 10 - in this case we have 3 conditions.
- smaller block has better coverage –very usual situation
- most concern to improve coverage has to be about condition coverage UNRs determined by FCA for blocks A and B

Table 2

FCA results

	Block A	**Block B**
Lines:		
found	123	7035
coverable	93	7035
uncoverable	30	0
Conditions:		
found	57	2377
coverable	42	131
uncoverable	15	2246
Signal bits:		
found	315	9013
coverable	269	8925
uncoverable	46	88
FSM states:		
found	1	3
coverable	1	3
uncoverable	0	0
FSM transitions:		
found	1	27
coverable	1	23
uncoverable	0	4
Total:		
found	509	18455
uncoverable	91 (18%)	2338 (12%)

Comments to the table above:

- most desired result of FCA is to get **uncoverable** constructs to exclude them from coverage report saving Designer and Verification Engineer time. As you can see relative number of UNR is not impressive, but look from other side: for block B we excluded 2338 coverage constructs!!! Plenty of manual analyzing time saved !
- Designer has to consider each uncoverable construct (especially line and toggle) as the potential source of Design redundancy
- FCA determined minimal amount of uncoverable constructs, because we did not constrain any Design inputs. In reality Design has some interfaces with specific protocols, which cause input ports dependencies, and some input combinations become impossible. It will potentially increase amount of UNRs, but requires additional manual work to be done by creating and debugging some constraints.

VI. Computational resources

FCA run was done locally (not on computer farm), on one Linux workstation, with 8 Xeon processors, each with 4 cores. Memory – 32GB.

FCA job used 1 processor with 4 cores, virtual memory – up to 17GB.

For block A elapsed time is ~ 1 day job run.

For block B ~ 7 days job run.

VII. Conclusions

- It is worth to use – remember: found 2338 uncoverable constructs for block B.
- Preparation for FCA run is very minimal ~ 10 min for me.
- FCA job is highly paralleled and running on network can shorten job time.
- This tool became part of the design and verification methodology within our organization following its successful evaluation.
- This presentation is first step in Formal methods use for verification. Next step has been done [3].

Acknowledgements

Author thanks my colleagues – Verification Engineers of company SK Hynix memory solutions (San Jose, CA, the USA), who reviewed previous versions of my presentation and applied this approach to get their blocks coverage improvements, sometimes getting even more impressive results.

References

[1] VC Formal Coverage Analyzer User Guide, Version K-2015.09, September 2015, Synopsys

[2] VC Formal Verification User Guide, Version K-2015.09, September 2015, Synopsys
[3] Tatarnikov Y., Labib K. Next step of Formal Verification utilization Available at https://www.synopsys.com/community/snug/snug-silicon-valley/location-proceedings-2018.html (accessed 03.05.2018).

УДК 519.714

Использование формального метода для улучшения покрытия проекта оцениваемого с помощью метрики «code coverage»

Ю.А. Татарников

Независимый контрактор, yuritatar@gmail.com

***Аннотация* — Данная презентация представляет результаты использования ППП «Формальный Анализатор» компании Синопсис. ППП основан на методах формальных доказательств.Определяются конструкции дизайна, представленного на регистровом уровне, которые не могут быть обнаружены любым тестом. Исключение этих конструкций улучшает показатели тестового покрытия дизайна. Анализ – составная часть технологии разработки логического дизайна СБИС**

***Ключевые слова* — СБИС, формальная верификация, моделирование, RTL, SystemVerilog, SVA.**

Литература

[1] VC Formal Coverage Analyzer User Guide, Version K-2015.09, September 2015, Synopsys.
[2] VC Formal Verification User Guide, Version K-2015.09, September 2015, Synopsys.
[3] Tatarnikov Y., Labib K. Next step of Formal Verification utilization. Available at https://www.synopsys.com/community/snug/snug-silicon-valley/location-proceedings-2018.html (accessed 03.05.2018).