


FPGA development meets IEC 62304

How to optimize and automate software lifecycle processes for FPGA based medical devices

Ingenieurbüro Tobias Baumann

www.elpra.de
tobias.baumann@elpra.de

September, 24th 2020

FPGA Verification Day 2020 presented by **TRIAS** 
mikroelektronik

About Me (really quick)

Short Vita

- Studied physics and microsystems engineering at Albert-Ludwigs University Freiburg
- Developed FPGA devices for IMTEK (Institut für Mikrosystemtechnik) and COMPASS Experiment at CERN
- Worked as FPGA development and verification engineer for a development service provider in medical and video processing
- Doing now the same as a self-employed and added DevOps engineering to my portfolio
- You're interested or want to know more? Just *mail*, *call* or *invite* me.

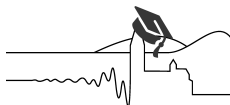


Table of Contents

1 Introduction

- What is this presentation about and what not?
- What is IEC 62304?

2 Verification strategies and requirements coverage

- Some example strategies from daily work
- Optimized verification strategies
- VUnit - A HDL testing framework
- Example Projects

3 Functional and Code Coverage

- General Stuff
- Modelsim / Questasim Example

4 Verification Environments (using Docker)

- Optimizing reproducibility for verification environments

5 Effective Software Configuration Management

- What is test-, build- automation?
- Using Gitlab as SCM tool

6 Summary and Book Recommendations

Table of Contents

1 Introduction

- What is this presentation about and what not?
- What is IEC 62304?

2 Verification strategies and requirements coverage

- Some example strategies from daily work
- Optimized verification strategies
- VUnit - A HDL testing framework
- Example Projects

3 Functional and Code Coverage

- General Stuff
- Modelsim / Questasim Example

4 Verification Environments (using Docker)

- Optimizing reproducibility for verification environments

5 Effective Software Configuration Management

- What is test-, build- automation?
- Using Gitlab as SCM tool

6 Summary and Book Recommendations

Introduction

What is this presentation about and what not?

- It's **not** about IEC 62304 itself
 - The IEC 62304 is large and complex. It's impossible to talk about it in detail in only 45 minutes.
 - The whole standard is too boring for FPGA developers - let project leaders deal with this, they get paid for it. ;-)
- It's about defining a modern and automated FPGA verification process and hopefully gives some ideas for a modern software stack.
 - And we will see how good it works with the IEC 62304.
 - But it's also a good process for everyone not in medical, everyone can learn from this presentation. :-)
- I'm not a IEC 62304 expert / auditor / consultant (remember: it's boring). It simply represents my experiences with my customers in medical industries.
- It focuses on beginners in FPGA verification automation - experience has shown that it is more probabilic to have beginners in a random sample of listeners than experts. If I'm wrong - I'm sorry!

Introduction

What is this presentation about and what not?

- Why can FPGA developers learn from it?
 - Most often FPGA developers are electrical engineers and have a poor education in modern software development techniques
⇒ (Yes, we all develop hardware but the design entry and techniques are the same than developing software ;-)).
 - How can externals like me help? Nearly every FPGA developer (I know) doesn't have any free time to
 - lay down work,
 - learn about process automation/optimization or *DevOps* principles
 - and establish modern development processes in their companies.
- What if I already know everything?
 - Enjoy the Buzzword bingo - maybe there is still a chance you will hear something new.

Introduction in IEC 62304

What is IEC 62304 in short?

- International standard which specifies **software lifecycle requirements** for developing medical products
- Standard is applicable on standalone medical software and **embedded medical software** (and FPGAs? - see next slide)
- It covers the **development** and maintenance process
- It **doesn't cover a specific** verification/validation and release process!
 - This does not mean you don't have to verify your software,
 - but it **doesn't tell you how** your software must be tested.
 - You must define verification strategies, evaluate them and document them in your software development plan.

Introduction in IEC 62304

Are FPGAs covered by IEC 62304?

Important question:

Is FPGA development hardware or software development?

Difficult question - easy answer:

- IEC 62304 only covers **software** lifecycle requirements!
- Let's assume that FPGA development is covered by IEC 62304 and auditors shall decide.



Introduction in IEC 62304

Relevant clauses for this presentation

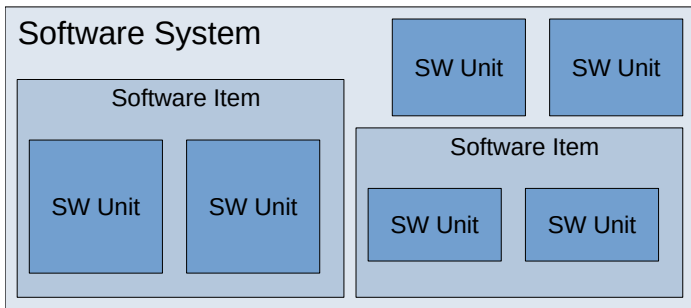
- 5 Software Development
 - 5.1 Development Planning
 - 5.2 Requirements Analysis
 - 5.3 Architectural Design
 - 5.4 Detailed Design
 - 5.5 Software Unit Implementation and Verification
 - 5.6 Software Integration and Integration Testing
 - 5.7 Software System Testing
 - 5.8 Release
- 6 Software Maintenance
- 7 Software Risk Management
- 8 Software Configuration Management

Note

Integration testing and system testing can be combined in one test plan (see clause 5.6.4).

Introduction in IEC 62304

Terminology



- Software architecture must be documented in design specification which describes the structure and identifies the Items / Units.
- Items / Units and their interfaces must be specified, documented (this are only a class C clauses) and verified.
- Safety Classes: Software System starts with class C, every Item / Unit can have its own class.

Introduction in IEC 62304

Terminology - Safety Classes

IEC 62304 Safety Classes

Class A No injury or damage to health possible.

Class B No serious injury possible.

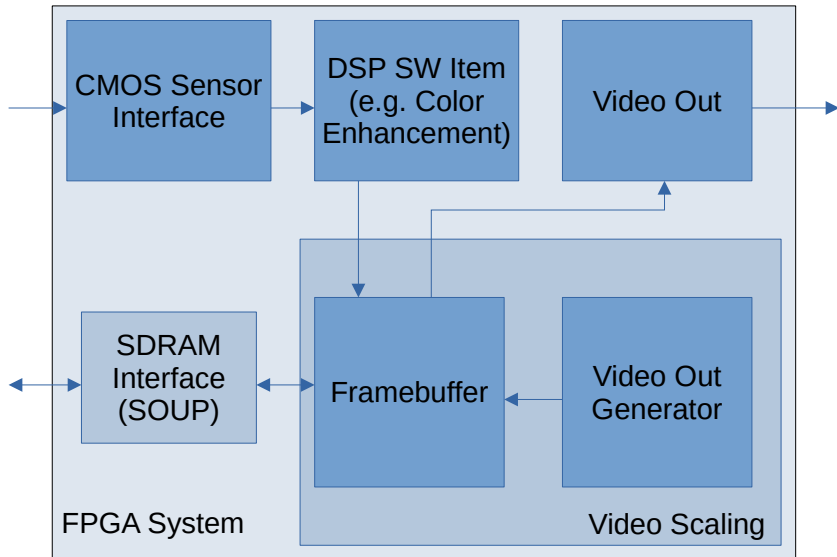
Class C Death or serious injury possible.

Notes

- The higher the class, the higher the amount of required documentation. (makes sense!)
- Safety class can be decreased by implementing hardware based risk control measures. (only one level possible!)

Introduction in IEC 62304

Terminology - Example FPGA Design



Introduction in IEC 62304

What do we have to verify (and document)?

	Focus on	Clause	Min. Safety Class
Software Implementation	Units	5.5.5	B
Software Integration	Items	5.6.2	B
Software Regression Tests	Items	5.6.2	B
Software System	System	5.7.x	B

Notes

- Documentation mentioned here is about the verification itself and the results, not the design!
- Integration and system can be done within a single test plan.

Table of Contents

1 Introduction

- What is this presentation about and what not?
- What is IEC 62304?

2 Verification strategies and requirements coverage

- Some example strategies from daily work
- Optimized verification strategies
- VUnit - A HDL testing framework
- Example Projects

3 Functional and Code Coverage

- General Stuff
- Modelsim / Questasim Example

4 Verification Environments (using Docker)

- Optimizing reproducibility for verification environments

5 Effective Software Configuration Management

- What is test-, build- automation?
- Using Gitlab as SCM tool

6 Summary and Book Recommendations

Verification strategies and requirements coverage

Some examples for software unit testing

Example:

Requirement A UART interface is provided between the FPGA and the microcontroller device.

Class C The patient can seriously be harmed if this UART interface fails.

Test plan The verification engineer picks an specific oscilloscope, makes specific settings, places the probes and verifies that the display shows the expected results.

Question:

Why is this a problem? It totally fulfils the IEC 62304 requirements if documented and evaluated correctly!

Verification strategies and requirements coverage

Some examples

Serious problems with this approach:

- Lets assume only a single char is transmitted via UART. How can functional coverage be guaranteed? Making 256 oscilloscope shots?
⇒ Lack of coverage can have a huge impact on safe operation!
- Using hardware for software verification
 - increases costs by buying hardware and measurement equipment.
 - increases costs by hiring engineers.
 - makes tracing test results difficult because a lot of office work is to do.
- Humans make errors - machines too but with way less probability!

Verification strategies and requirements coverage

Improvements on the example

Example: Better test strategy

Test plan A verification engineer runs a **self-checking testbench**, which generates a set of stimulus and checks the UUT output, to verify the software unit.

Is everything fine now?

This approach is way better, but not perfect!

- Functional coverage can be much improved (e.g. by coding a stimulus generator and checker or using state-of-the-art verification components),
- but still a lot manual work to do (run testbench, create test report, ...).

Verification strategies and requirements coverage

Final improvements (for nor) on the example

Example: Pretty good test strategy

Test plan A self-checking testbench within a **testing framework** is invoked for every test automatically after every commit (into the software configuration management system).

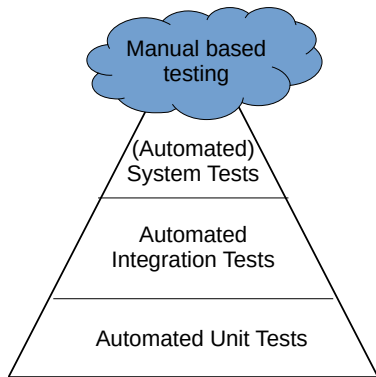
Is everything fine now?

Some details can be improved, but very good for the moment. :-)

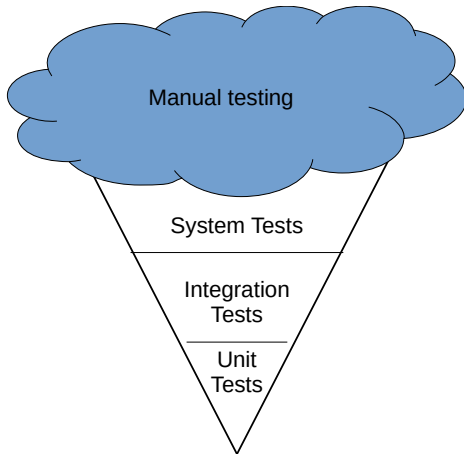
- Run tests with every commit (e.g. using Git Hooks, Gitlab CI, Github Actions, ...)
⇒ **“Test early and often strategy”** (find bugs as early as possible)
- Use tools to automatically generate test reports (e.g. using Python packages *reportlab*, *xlsxwriter*, ...)

Optimized verification strategies

Ideal vs non ideal testing pyramid



The ideal testing automation pyramid



The non-ideal testing automation pyramid

VUnit - A HDL testing framework

Introduction - What is VUnit?

What is VUnit?

- An open source unit testing framework for VHDL/SystemVerilog.
- Python test suite runner that enables powerful test administration, can continue testing after fatal run-time errors (e.g. division by zero), and ensures test case independence.
- Automatic scanning of files for tests, file dependencies, and file changes enable automatic (re)compilation and execution of test suites.
- Support for running test benches with multiple generic/parameter settings.
- Assertion checker library that extends VHDL built-in support (assert).
- Requirements trace-ability through JSON Export and test attributes.
- Outputs JUnit report files for Gitlab (or Jenkins) integration.

VUnit - A HDL testing framework

Introduction - Why using VUnit?

Why using VUnit?

- Open Source (Mozilla Public License, v. 2.0)
- It is Python based. You can ...
 - create test reports as PDF, Excel Sheet and many other formats.
 - easily create stimuli and test vectors (e.g. write and examine a video algorithm in Python and use this piece of code as golden model for your testbench).
 - do everything that Python can do - Python is also easy to learn and very powerful!
- Many simulation tools are supported, e.g. Modelsim / Questasim and the open source simulator GHDL.
- VUnit developers (Lars Asplund, et al.) are very active and supportive. Follow the project on Github: <https://github.com/VUnit/vunit>

VUnit - A HDL testing framework

Introduction - How does VUnit work?

How does VUnit work?

- ➊ Create a Python script as runner (e.g. run.py) and instantiate VUnit.
- ➋ Specify UUT / testbench HDL files and associate them to a library.
- ➌ Set tests and their configurations (e.g. multiple generics settings).
- ➍ Add additional Python code, e.g. for ...
 - creating stimulus from golden models.
 - creating requirement coverage reports.
 - A lot more - be creative, there's nothing Python can't do and nearly every problem is solved by a specific Python package!
- ➎ Run the Python runner script from command line (maybe with specific arguments) to start compilation and simulation.
- ➏ See the results on command line and archive the generated artefacts.

(Let's see an example to demonstrate.)

VUnit - A HDL testing framework

Example Project

Example project

<https://gitlab.com/Elpra/fpga-verification-days-2020-examples/vunit-example>

```
Compiling into vunit_lib: ../../../../local/lib/python3.8/site-packages/vunit/vhdl/check/src/check_deprecated_pkg.vhd      passed
Compiling into vunit_lib: ../../../../local/lib/python3.8/site-packages/vunit/vhdl/vunit_context.vhd                passed
Compiling into tb_lib:    tests/tb_uut.vhd                                                                passed
Compiling into vunit_lib: ../../../../local/lib/python3.8/site-packages/vunit/vhdl/check/src/check.vhd                passed
Compile passed

Starting tb_lib.tb_uut.Data Width 4
Output file: vunit_out/test_output/tb_lib.tb_uut.Data_Width_4_d625945e741537b5c28feefdc405a28101816512/output.txt
pass (P=1 S=0 F=0 T=4) tb_lib.tb_uut.Data Width 4 (0.7 seconds)

Starting tb_lib.tb_uut.Data Width 8
Output file: vunit_out/test_output/tb_lib.tb_uut.Data_Width_8_38fc82ee3ac37fed7b5c9d53c6addf8e74d61881/output.txt
pass (P=2 S=0 F=0 T=4) tb_lib.tb_uut.Data Width 8 (0.2 seconds)

Starting tb_lib.tb_uut.Data Width 16
Output file: vunit_out/test_output/tb_lib.tb_uut.Data_Width_16_d8e9544e780ff0726464cadf0beab97ce1ba2cc6/output.txt
pass (P=3 S=0 F=0 T=4) tb_lib.tb_uut.Data Width 16 (1.0 seconds)

Starting tb_lib.tb_uut.Data Width 32
Output file: vunit_out/test_output/tb_lib.tb_uut.Data_Width_32_a7e1cacff24fd36d7d893d2ba7a5323642b29a73/output.txt
pass (P=4 S=0 F=0 T=4) tb_lib.tb_uut.Data Width 32 (0.1 seconds)

==== Summary =====
pass tb_lib.tb_uut.Data Width 4 (0.7 seconds)
pass tb_lib.tb_uut.Data Width 8 (0.2 seconds)
pass tb_lib.tb_uut.Data Width 16 (1.0 seconds)
pass tb_lib.tb_uut.Data Width 32 (0.1 seconds)
=====
pass 4 of 4
=====
Total time was 2.0 seconds
Elapsed time was 2.0 seconds
=====
All passed!
[baumant@devel-laptop vunit-example]$
```

A Real Example

A little overview

Folder structure

- bin - Contains start and document creation script.
- src - Contains the UUT sources and patches (based on schematic design entry).
- tests - Contains configuration YAML data and Python helper scripts.
- tests/libs - Contains simulation models for external devices
- tests/model_tests - Contains VHDL tbs and run.py for model verification.
- tests/sub_tests - Contains VHDL tbs and run.py for SW Units verification.
- tests/system_tests - Contains VHDL tbs and run.py for SW Items integration testing (post-synthesis simulation).

A Real Example

Verification flow - completely automated with every commit

- ❶ Preparation step - just to have a chance to simulate the horror
 - ❶ Apply patches using `git apply`.
 - ❷ Create a post-synthesis VHDL model using a set of Quartus CLI tools.
 - ❸ Translate schematics to VHDL files using `quartus_map`.
- ❷ Execute VUnit runners in `model_tests`, `sub_tests` and `sub_tests`.
 - ❶ Read configuration YAML
 - ❷ Create set of VHDL dependencies and compile them
 - ❸ Create simulation configuration
 - ❹ Run simulation
 - ❺ Repeat this for every runner
- ❸ Create test report from verification results / artefacts.

Automation with every commit

We will see later how this is automated with Docker containers.

A Real Example

YAML configuration file (part 1)

```
1  ▼ test types:
2    - simulation model # Verifies simulation models
3    - submodule       # VHDL simulation from netlist conversion
4    - system          # Post synthesis simulation
5
6  ▼ requirements:
7    P12345678: # Software System ID
8      SRS0001: accepted # Here could also be more information about the requirement, to generate the documents automatically
9      SRS0002: rejected
10     SRS0003: accepted
11     SRS0004: accepted
12
13 ▼ simulation models:
14   - Proprietary UART
15   - SRAM
16   - ADC
17
18 ▼ tb_sram:
19   id: 1
20   test type: simulation model
21   name: "SRAM Interface"
22   description: "Verifies the simulation model for SRAM interface."
23   ▼ req coverage:
24     P12345678: {}
25   sim coverage: ["SRAM"]
26   generics: {}
27   sim options: {}
28
29 ▼ tb_uart:
30   id: 2
31   test type: simulation model
32   name: "Proprietary UART"
33   description: "Verifies the simulation models for the UART interface."
34   ▼ req coverage:
35     P12345678: {}
36   sim coverage: ["Proprietary UART", "SRAM"]
37   generics: {}
38   sim options: {}
```

A Real Example

YAML configuration file (part 2)

```
40 ▼ tb_adc:
41   id: 3
42   test type: simulation model
43   name: "ADC Model"
44   description: "Model of ADC with serial interface."
45   ▼ req coverage:
46     P12345678: {}
47   sim coverage: ["ADC"]
48   generics: {}
49   sim options: {}
50
51 ▼ tb_fpga_version_readout:
52   id: 4
53   test type: system
54   name: "SRAM Interface - FPGA Version Readout"
55   description: "Verifies that FPGA version and revision can be read out correctly."
56   ▼ req coverage:
57     P12345678: ["SRS0001", "SRS0003"]
58   sim coverage: ["SRAM"]
59   ▼ generics:
60     FPGA_VERSION: 2
61     FPGA_REVISION: 1
62   sim options: {}
63
64 ▼ tb_adc_dsp:
65   id: 5
66   test type: submodule
67   name: "ADC Data Processing"
68   description: "Verifies that ADC data processing works as expected."
69   ▼ req coverage:
70     P12345678: ["SRS0003", "SRS0004"]
71   sim coverage: ["ADC"]
72   generics: {}
73   sim options: {}
```

A Real Example

The core of the runner

```
1 | def create_test_configs(vu, test_type, test_folder):
2 |
3 |     tb_lib = vu.add_library("tb_lib")
4 |     tb_lib.add_source_files("tests/*.vhd")
5 |     tb_lib.add_source_files("tests/" + test_folder + "/*.vhd")
6 |
7 |     TEST_CONFIG_YAML_PATH = 'tests/test_config.yaml'
8 |
9 |     with open(TEST_CONFIG_YAML_PATH, "r") as read_file:
10 |         yaml_objects = yaml.load_all(read_file, Loader=yaml.FullLoader)
11 |         testbenches = []
12 |
13 |         for yaml_object in yaml_objects:
14 |             for yaml_name, yaml_params in yaml_object.items():
15 |
16 |                 if yaml_name.lower() not in (tb.name for tb in tb_lib.get_test_benches()):
17 |                     continue
18 |
19 |                 testbench_name = yaml_name
20 |                 config_params = yaml_params
21 |
22 |                 assert config_params["test type"] == test_type, "Wrong type of test!"
23 |
24 |                 testbenches.append(tb_lib.entity(testbench_name))
25 |                 testbench = testbenches[-1]
26 |
27 |                 # Check requirements integrity for items
28 |                 for item in config_params["req coverage"]:
29 |                     text = "Requirement error in " + testbench_name + " for " + item
30 |                     assert set(config_params["req coverage"][item]).issubset(requirements[item]), text
31 |
32 |                 # Check requirements integrity for simulation models
33 |                 text = "Requirement error in " + testbench_name + " for simulation models"
34 |                 assert set(config_params["sim coverage"]).issubset(simulation_models), text
35 |
36 |                 attributes = {}
37 |                 for attr in ("id", "description", "req coverage", "sim coverage"):
38 |                     attributes["." + attr] = config_params[attr]
39 |
40 |                 generics = config_params["generics"]
41 |                 all_srs = []
42 |
43 |                 if "P12345678" in config_params["req coverage"]:
44 |                     for srs in config_params["req coverage"]["P12345678"]:
45 |                         all_srs.append(str(int(srs[-4:])))
46 |                     if len(all_srs) > 0:
47 |                         generics["REQS_LIST"] = ','.join(all_srs)
48 |
49 |                 testbench.add_config(
50 |                     name=config_params["name"],
51 |                     generics=generics,
52 |                     attributes=attributes
53 |                 )
```

Part of the test report (coverage table)

Requirements Coverage

P14000111900004	accepted	
P14000111900005	accepted	A. Shilovsk, Jan, with support
P14000111900006	accepted	A. Shilovsk, Jan
P14000111900007	accepted	A. Shilovsk, Jan
P14000111900008	accepted	A. Shilovsk, Jan
P14000111900009	accepted	A. Shilovsk, Jan
P14000111900010	accepted	A. Shilovsk, Jan
P14000111900011	accepted	A. Shilovsk, Jan
P14000111900012	accepted	A. Shilovsk, Jan
P14000111900013	accepted	A. Shilovsk, Jan
P14000111900014	accepted	A. Shilovsk, Jan
P14000111900015	accepted	A. Shilovsk, Jan
P14000111900016	accepted	A. Shilovsk, Jan
P14000111900017	accepted	A. Shilovsk, Jan
P14000111900018	accepted	A. Shilovsk, Jan
P14000111900019	accepted	A. Shilovsk, Jan
P14000111900020	accepted	A. Shilovsk, Jan, with support
P14000111900021	accepted	
P14000111900022	accepted	
P14000111900023	accepted	A. Shilovsk, Jan, with support
P14000111900024	accepted	A. Shilovsk, Jan, with support
P14000111900025	accepted	A. Shilovsk, Jan, with support
P14000111900026	accepted	A. Shilovsk, Jan, with support
P14000111900027	accepted	A. Shilovsk, Jan, with support
P14000111900028	accepted	A. Shilovsk, Jan, with support
P14000111900029	accepted	A. Shilovsk, Jan
P14000111900030	accepted	A. Shilovsk, Jan

Verification Results

[illegible]

Table of Contents

- 1 Introduction
 - What is this presentation about and what not?
 - What is IEC 62304?
- 2 Verification strategies and requirements coverage
 - Some example strategies from daily work
 - Optimized verification strategies
 - VUnit - A HDL testing framework
 - Example Projects
- 3 Functional and Code Coverage
 - General Stuff
 - Modelsim / Questasim Example
- 4 Verification Environments (using Docker)
 - Optimizing reproducibility for verification environments
- 5 Effective Software Configuration Management
 - What is test-, build- automation?
 - Using Gitlab as SCM tool
- 6 Summary and Book Recommendations

Functional and Code Coverage

Defining Software Unit Acceptance Criteria

Define acceptance criterias - when is verification done?

IEC 62304 prescribes the definition of acceptance criterias. These need to be documented within software development plan (clause 5.1.6).

Some important acceptance criteria

- **Design Guidelines** (maybe technology dependend, e.g. reset handling, clock-domain-crossing!)
- Coding Guidelines - see clause 5.5.3
- Static Code Analysis (Linting) - see clause 5.5.4 (additional acceptance criterias)
- **Functional Coverage** - see clause 5.5.2
- **Code Coverage** - see none normative part of IEC 62304:2015
- ...

Functional and Code Coverage

Excource: Acceptance criteria checking automation

Also the other criterias can be automated!

A lot tools are available for the other points, e.g.:

- See *Questa Formal Solutions & Apps*, e.g. for clock domain crossing
- *Code Climate* a coding review and statical analysis tool (see <https://codeclimate.com> - for VHDL / Verilog a own and maybe technology dependend engine needs to be developed!)
- Synthesis and P&R messages can also help to find problems in a early development state.
- ...

And even the System Test can be automated ...

... by using a hardware-in-the-loop verification environment. (Very expensive but very effective verification environment!)

Functional Coverage

What is functional coverage? (very quick - we all know it)

Functional Coverage

is a metric which shows **how much of our functionality** has been covered by our verification environment.

Our counter example from previous section

- Waiting for 5 cycles after clock enabling \Rightarrow would cover only 5 output states of the counter ($\approx 2\%$ for 8 bit counter).
- Assuming 64 bit counter: Simulation would never end (maybe think about formal verification).

Define acceptance criterias for functional coverage

Define meaningful metrics and argue why they make your product safe.

Code Coverage

What is code coverage? (also very quick - we even know this)

Code Coverage

is a metric which shows **how much of our source code** has been executed by our verification environment.

Most important code coverage types in IEC 62304

Statement Analyzes that every line of code is executed at least once.

Some Pseudo-Code: if (a or b) then x := 0; else x := 1; endif;

Branch Analyzes that every branch is executed at least once (e.g. a sticks to true and b is varied).

Condition Analyzes decisions made in “if”. Every condition expression is at least once true and false (a is varied and b is varied).

Functional and Code Coverage

When is testing complete and done?

Simple answer:

When for every data signal every state combination and permutation of transistions are simulated and verified.

- Not possible even in smaller designs!
- Formal Verification tools can help, but aren't enough we can do.

An IEC 62304 related answer (with examples)

We need to specify criterias and argue within our risk management why we go our way (remember: IEC 62304 doesn't tell us how we have to verify!)

- Defining Golden Models from where we generate testvectors from and specify against them.
- Creating random stimulus and defining coverage metrics.
- This can pretty good be combined with assertion-based verification!
- Using verification IPs which are already evaluated and verified.

Functional Coverage

Tools Summary

Open Source Tools (those I know and use)

- OSVVM (Open Source VHDL Verification Methodology - every FPGA verification engineer should know)
- UVVM (Universal VHDL Verification Methodology - the real expert is here ;-))
- VUnit (includes some verification IPs and extended VHDL assertions)

Commercial Tools

- Mentors offers a lot of stuff as Trias / Mentor for more information.
 - PSL, Verification IPs, Formal Verification Apps, ...
 - UVM (as UVVM but for SystemVerilog)
 - OVM (Open Verification Methodology - never used it)
 - very much more ..., ask Trias / Mentor for more information
- Never used other vendors, but I'm sure there are a lot good tools on the market.

Code Coverage

What does the IEC 62304 require?

IEC 62304 only gives a recommendation.

Min. Safety Class	Code Coverage Type
A	Statement Coverage
B	Branch Coverage
C	Modified Condition/Decision Coverage

My personal recommendation for a verification process

- Code coverage can help to make software safer and are state-of-the-art in modern software engineering.
- Define metrics that make sense, e.g. coverage percentage must be increased with every iteration. Decreasing must be argued with a really important reason!
- Creating code coverage metrics are very easy, there is no reason to avoid!

Modelsim / Questasim Example

Hits	BC	Ln#	
		6	generic(
		7	DATA_WIDTH : positive
		8);
		9	port(
		10	clk_in : in std_logic;
		11	ce_in : in std_logic;
		12	data_q : out std_logic_vector(DATA_WIDTH-1 downto 0)
		13);
		14	end entity uut;
		15	
		16	architecture RTL of uut is
		17	signal data : unsigned(data_q'range) := (others => '0');
		18	begin
		19	
✓		20	data_q <= std_logic_vector(data);
		21	
		22	ctr_proc : process (clk_in)
		23	begin
✓		24	if rising_edge(clk_in) then
X	Xr	25	if (ce_in = '0') then
✓		26	data <= (others => '0');
✓		27	else
		28	data <= data + 1;
		29	end if;
		30	end if;
		31	end process;
		32	
		33	end architecture RTL;
		34	
		35	

Analysis x Cover Directives x Covergroups x uut.vhd x

Coverage Details

By file
File: src/uut.vhd
Line: 24
Branch Coverage for:
if rising_edge(clk_in)
Active: 258, True Hits: 258, AllFalse: Auto Excluded
* Excluded because the clkOpt optimization has removed this branch

Modelsim / Questasim Example



Testplan Design DesUnits

tb_uut
uut

Questa Design Coverage

Scope: [/tb_uut](#)

Instance Path:
[/tb_uut](#)

Design Unit Name:
[work.tb_uut\(rtl\)](#)

Language:
VHDL

Source File:
[tests/tb_uut.vhd](#)

Coverage Summary By Instance:

Scope	TOTAL	Statement	Branch	Toggle	Assertion
TOTAL	79.38	66.66	66.66	85.00	100.00
tb_uut	100.00	--	--	--	100.00
uut	72.77	66.66	66.66	85.00	--

Local Instance Coverage Details:

Total Coverage:					100.00%	100.00%
Coverage Type	Bins	Hits	Misses	Weight	% Hit	Coverage
Assertions	1	1	0	1	100.00%	100.00%

Recursive Hierarchical Coverage Details:

Total Coverage:					82.97%	79.58%
Coverage Type	Bins	Hits	Misses	Weight	% Hit	Coverage
Statements	3	2	1	1	66.66%	66.66%
Branches	3	2	1	1	66.66%	66.66%
Toggles	40	34	6	1	85.00%	85.00%
Assertions	1	1	0	1	100.00%	100.00%

Coverage Report Summary Data by file

File: src/uut.vhd

Enabled Coverage	Active	Hits	Misses	% Covered
-----	-----	----	-----	-----
Stmts	3	2	1	66.66
Branches	3	2	1	66.66
Toggle Bins	40	34	6	85.00

TOTAL ASSERTION COVERAGE: 100.00% ASSERTIONS: 1

Total Coverage By File (code coverage only, filtered view): 72.77%

Modelsim / Questasim Example

Little modification - much effort

Without Coverage

```
vlib uut_lib
vlib tb_lib
vmap work tb_lib
vcom -2008 -work uut_lib {src/uut.vhd}
vcom -2008 -work work {tests/tb_uut.vhd}
vsim tb_uut
run -all
```

With Coverage

```
vlib uut_lib
vlib tb_lib
vmap work tb_lib
vcom -2008 -work uut_lib +cover=bcesxf {src/uut.vhd}
vcom -2008 -work work {tests/tb_uut.vhd}
vsim -coverage tb_uut
run -all
```

Table of Contents

- 1 Introduction
 - What is this presentation about and what not?
 - What is IEC 62304?
- 2 Verification strategies and requirements coverage
 - Some example strategies from daily work
 - Optimized verification strategies
 - VUnit - A HDL testing framework
 - Example Projects
- 3 Functional and Code Coverage
 - General Stuff
 - Modelsim / Questasim Example
- 4 Verification Environments (using Docker)
 - Optimizing reproducibility for verification environments
- 5 Effective Software Configuration Management
 - What is test-, build- automation?
 - Using Gitlab as SCM tool
- 6 Summary and Book Recommendations

Verification environments

Reproducibility - What does IEC 62304 require?

Clause 5.6.7 - Integration test record contents

- We must keep sufficient documents to allow the test to be repeated.
- This can be documents about verification environments (including software tools) which were used for the verification.
 - Simulator name and version
 - Version of IPs, e.g. FPGA primitive simulation models
 - Test vector generators?

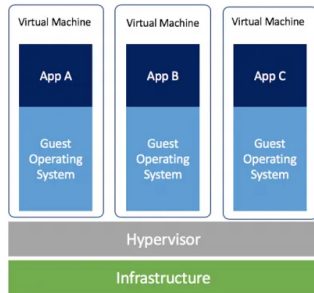
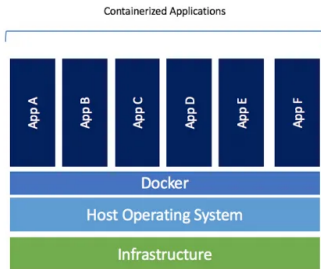
Clause 5.8.5 - Documentation of the build process

- We must describe the build process. Which tools were used and how where they invoked?
 - FPGA vendor tools for synthesis, Place & Route.
 - Additional tools for generating the bitstream.
 - Everything else what was used to create the release.
- The release process must also be reproducible!

Verification environments

What is Docker?

- Free software to isolate applications within a container
- Containers are isolated from one and each another and bundle their own software, libraries and configuration files.
- All containers are run by a single operating system kernel and therefore use fewer resources than virtual machines.
- Creation of container image is based on a *Dockerfile*.



Verification environments

How does it help?

- The Dockerfile completely specifies what components are within a Docker image.
- The Dockerfile is easy to track and to version - it's a simple text file. (Think about a problem occurring in the field, you simply can go back to every development / verification environment).
- Your verification environment needs to be evaluated and qualified. Using Docker you can automate this process!
- When a image is created, the applications within can be used on every physical machine where Docker is available. It doesn't depend on your active development machine. \Rightarrow Think about FPGA vendor tools which are only running on Win XP machines!
- Virtual machines can do this too? Yes, but not even close that productive.
- If you want to get rid of your powerful local machines (e.g. for large synthesis jobs), you can easily move those processes into the cloud.

Docker Example

Example - Create a container with Altera and Modelsim

```
1 ARG INSTALL_DIR=/opt/altera/13.0sp1
2
3 FROM centos:7 AS base
4 LABEL maintainer="Tobias Baumann <tobias.baumann@elpra.de>"
5 RUN yum -y update && \
6     yum install -y \
7     \
8     \
9     \
10    \
11    yum clean all
12
13 FROM base AS quartus-build
14 ARG INSTALL_DIR
15 ADD Quartus-web-13.0.1.232-linux.tar /home
16 WORKDIR /home
17 RUN ./setup.sh --mode unattended --unattendedmodeui minimal --installdir ${INSTALL_DIR} --disable-components quartus_help,modelsim_ae,max_web,devinfo,arria_web,cyclonev && \
18     sed -i 's,linux_rh60,linux,g' /opt/altera/13.0sp1/modelsim_ase/vco
19 RUN du -sh ${INSTALL_DIR}
20
21
22 FROM base
23 ARG INSTALL_DIR
24 COPY --from=quartus-build ${INSTALL_DIR} ${INSTALL_DIR}
25 RUN yum install -y python3 python3-pip && yum clean all
26 ENV PATH="${INSTALL_DIR}/quartus/bin:${INSTALL_DIR}/modelsim_ase/bin:${PATH}"
27 RUN pip3 install vunit-hdl==4.4.0 PyYAML
```

Example can be found under

<https://gitlab.com/Elpra/fpga-verification-days-2020-examples/docker-example>

Docker Example

Example - Create a container

Create, tag and push a image

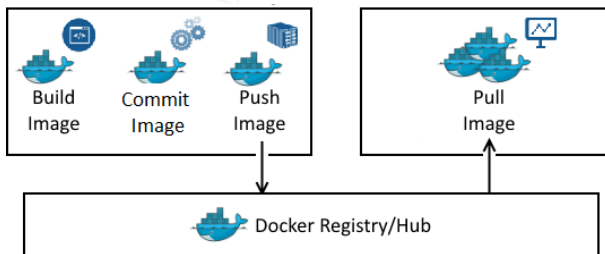
(run in same directory as Dockerfile)

```
docker build -t altera:v.1.2.3-latest .
```

```
docker push altera:v.1.2.3-latest
```

Default Docker Repository

Images are pushed/pulled to/from *Docker Hub* by default. You can use a selfhosted repository, e.g. within Gitlab, to keep your images inhouse.



Docker Example




Example - Pull a container

Pull specific version from repository

```
docker pull altera:v.1.2.3-latest
```

Tags

This repository contains 5 tag(s).

latest		🕒 8 months ago
2019.1.0		🕒 8 months ago
2018.2.1		🕒 2 years ago
2018.2.0		🕒 2 years ago
2018.1.0		🕒 2 years ago

Docker Example

Example - Run a application within a container

Call Docker

```
docker run -v $(pwd):/project altera:1.1.0 /project/bin/run_tests.sh
```

- Creates a new container from image altera:1.1.0.
- Mounts UUT and testbench sources into container folder /home/project.
- Runs the run_tests.sh script within the container.

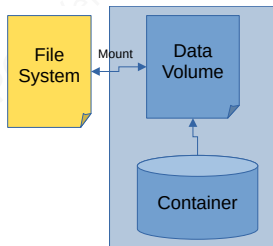


Table of Contents

1 Introduction

- What is this presentation about and what not?
- What is IEC 62304?

2 Verification strategies and requirements coverage

- Some example strategies from daily work
- Optimized verification strategies
- VUnit - A HDL testing framework
- Example Projects

3 Functional and Code Coverage

- General Stuff
- Modelsim / Questasim Example

4 Verification Environments (using Docker)

- Optimizing reproducibility for verification environments

5 Effective Software Configuration Management

- What is test-, build- automation?
- Using Gitlab as SCM tool

6 Summary and Book Recommendations

Effective Software Configuration Management

What is build automation?

What is build automation?

Build automation is the process of automating the creation of a software build and the associated processes. Including:

- Running automated tests / **verification automation**
- Source code compiling
- Packaging and releasing

Famous build automation servers available:

- **Gitlab CI**
- Jenkins / Hudson
- Github Actions & Travis CI

My favorite is ...

... **Gitlab** - but every tool is better than avoiding build automation.

Effective Software Configuration Management

Build automation with Gitlab CI - A mighty Tool!

What is Gitlab CI?

- It's part of Gitlab which manages projects builds / verification / release jobs and provides a nice user interface.
- Gitlab Runners are decentralized applications which are polling Gitlab CI via its API to check periodically if there is a new job available.
- If a new job is available, the Runner checks out the code and runs the given jobs in a defined pipeline.
- If a job has finished, job artefacts are uploaded and are available within Gitlab CI.
- The CI configuration is part of the repository, described in `.gitlab-ci.yml` and therefore versioned within the development process. That is what IEC 62304 wants from you. ;-)



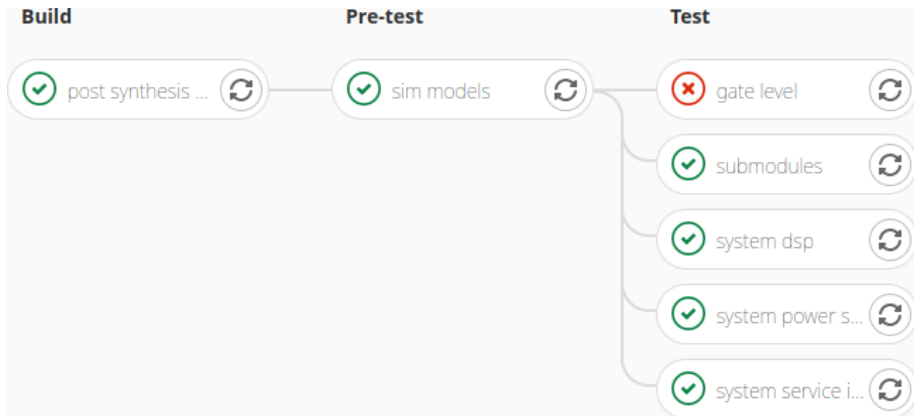
Effective Software Configuration Management

.gitlab-ci.yml Example

```
1 stages:
2   - build
3   - pre-test
4   - test
5
6 post synthesis model:
7   stage: build
8   image: registry.elpra.dev/development/docker-repo/quartus:13.0sp1-1.0.0
9   before_script:
10    - yum install -y git
11   script:
12    - bin/prepare
13   artifacts:
14     expire_in: 2 days
15     when: always
16     paths:
17     - some artefacts
18
19 sim models: &test-template
20   stage: pre-test
21   image: elpra/ghdl:altera
22   needs: []
23   variables:
24     RUN_SCRIPT: tests/model_tests/run.py
25   script:
26     - python3 $RUN_SCRIPT --export-json results.json
27     - python3 $RUN_SCRIPT --clean -v -x results.xml
28   artifacts:
29     expire_in: 2 days
30     when: always
31   reports:
32     junit:
33     - ./results.xml
```

Effective Software Configuration Management

Gitlab CI pipeline



Effective Software Configuration Management

Gitlab CI pipeline - Example

Our counter example in a Gitlab CI process ...

[https://gitlab.com/Elpra/fpga-verification-days-2020-examples/
gitlab-ci-example](https://gitlab.com/Elpra/fpga-verification-days-2020-examples/gitlab-ci-example)

Table of Contents

- 1 Introduction
 - What is this presentation about and what not?
 - What is IEC 62304?
- 2 Verification strategies and requirements coverage
 - Some example strategies from daily work
 - Optimized verification strategies
 - VUnit - A HDL testing framework
 - Example Projects
- 3 Functional and Code Coverage
 - General Stuff
 - Modelsim / Questasim Example
- 4 Verification Environments (using Docker)
 - Optimizing reproducibility for verification environments
- 5 Effective Software Configuration Management
 - What is test-, build- automation?
 - Using Gitlab as SCM tool
- 6 Summary and Book Recommendations

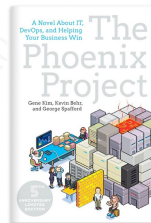
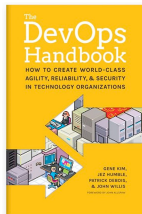
Summary

What have we learned?

- A very, very little about IEC 62304.
⇒ The whole standard is pretty complex. If you are further interested, maybe a training at <https://www.johner-institut.de/> is the right thing for you.
- VUnit is a great testing framework to manage HDL testbenches. We also saw that Python (which VUnit is based on) is generally a good idea for test automation and requirements coverage (which is prescribed by IEC 62304)
- Functional and code coverage helps us to define acceptance criteria which we need to define our verification goals.
- Docker helps us to create reproducible development, verification and release environments.
- To unify all this great tools we should automate them within our SCM (e.g. Gitlab). Our longterm target should be: Create an environment where we get a deliverable product with every commit (including documentation!).

Book Recommendations

Some further literature I can really recommend



- “*Basiswissen Medizinische Software*” (Johner et al.): Great book when starting to deal with medical software engineering and project leading.
- “*The DevOps Handbook*” (Kim et al.): The standard piece of literature for DevOps principles which I can highly recommend.
- “*The Phoenix Project*” (Kim et al.): Entertainment and education - it's a **novel** and a pleasure to read. Ideally read it before looking into *The DevOps Handbook*.

Thanks for your Attention

Let me know if you ...

- ... have any more questions (please use the Q&A Session).
- ... liked the presentation and want to dig deeper.
- ... disliked the presentation. Any feedback is appreciated.