EmLogic

Get the right FPGA quality through Efficient Verification and Requirements Tracking

SEFUW 2025, ESTEC

(by Espen Tallaksen, CEO EmLogic)

- Independent Design Centre for Embedded Systems and FPGA
- Established 1st of January 2021. **Extreme ramp up**
  - January 2021: 1 person
  - March 2025: → 48 persons (SW, HW, FPGA:21, DSP)
- Continues the legacy from bitvis
  - All previous Bitvis technical managers are now in EmLogic
- Verification IP and Methodology provider **UVVM**
- Course provider within FPGA Design and Verification
  - Accelerating FPGA Design (Architecture, Clocking, Timing, Coding, Quality, Design for Reuse, …)
  - Advanced VHDL Verification – Made simple (Modern efficient verification using UVVM)
- A potential partner for ESA projects for European companies
  - Increased opportunities due to Norway's low geo return

What is UVVM?

UVVM = Universal VHDL Verification Methodology

- VHDL Verification Library & Methodology
- Free and Open Source
- Used by >35% of all FPGA VHDL designers in Europe

- Results in a very structured testbench architecture
- Significantly improves Verification Efficiency
- Assures a far better Design Quality

- Recommended by Doulos for Testbench architecture
- ESA projects to extend the functionality
- IEEE Standards Association Open source project

- Runs on any VHDL-2008 compliant simulator

The ESA projects

- ESA UVVM projects were initiated in order to:
  - Solve challenges on verification of FPGAs and IP
  - Provide the best possible VHDL verification methodology
    - For both suppliers to ESA – and the FPGA/VHDL community in general

ESA UVVM 1: 2017-2019

- Scoreboards
- Monitors
- Error injection
- Local sequencers
- Transaction info
- Watchdogs
- Hierarchical VVCs
- Specification Coverage

ESA UVVM 2: 2020-2022

- Enhanced Randomisation
- Optimised Randomisation
- Functional Coverage
- Extensions

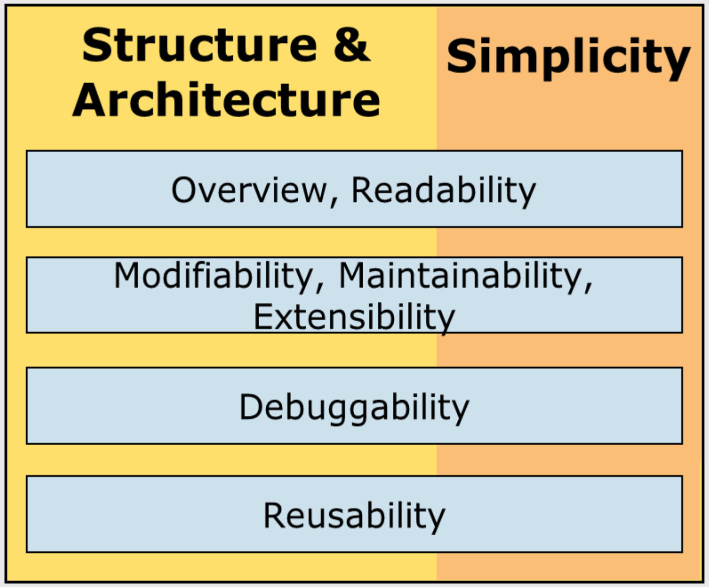
ESA UVVM 3: 2024-2025

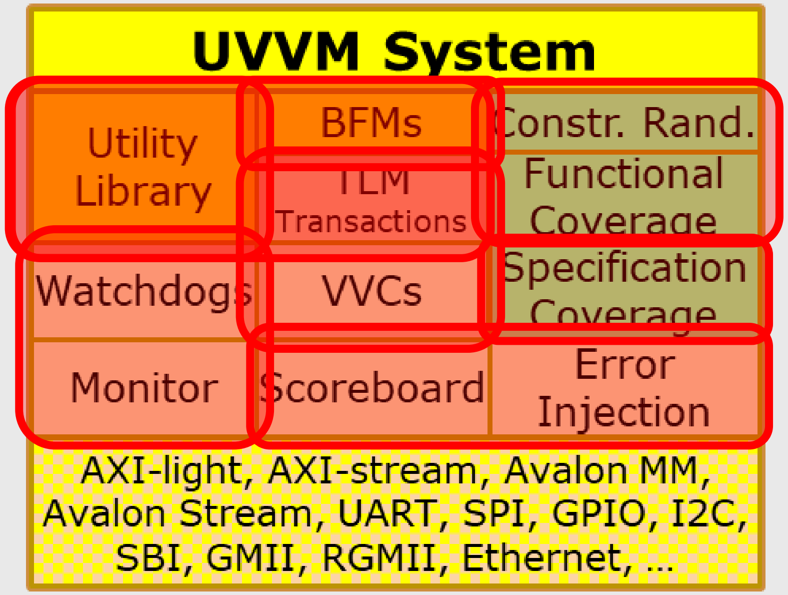
- Completion detection
- Detection of unwanted interface activity
- SV-extended Randomisation
- More to be announced

In parallel with "normal" extensions and maintenance

What enables Quality and Efficiency

- Huge improvement potential for more structured FPGA verification

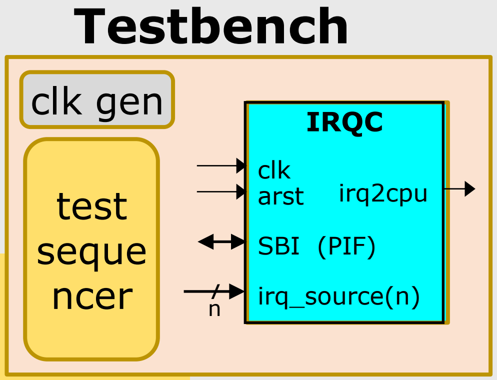




UVVM targets all of this

EmLogic

Example on test sequencer code and transcript/log



```
clock_generator(clk, GC_CLK_PERIOD);
```

```
log(ID_LOG_HDR, "Check Interrupt trigger clear mechanism");
check_value(irq2cpu, '0', "irq2cpu default inactive");
check_stable(irq2cpu, now – v_reset_time, "Stable irq2cpu");
gen_pulse(irq_source(3), '1', C_CLK_PERIOD, "Set IRQ source 3 for clock period");
await_value(irq2cpu, '1', 0 ns, 2* C_CLK_PERIOD, "Interrupt expected");
sbi_write(C_ADDR_ITR, x"AA", "ITR : Set interrupts");
```

```
2000.0 ns     Check Interrupt trigger clear mechanism
----------------------------------------------------------------------
 110.0 ns     check_value() => OK, for std_logic '0'. irq2cpu default
 727.5 ns     check_stable() => OK. Stable at 0. Stable irq2cpu
1060.0 ns     Pulsed to '1'. Set IRQ source 3 for clock period
1117.5 ns     await_value(std_logic 1, 0 ns, 20 ns) => OK. Interrupt expected
2020.0 ns     SBI write(A:x"2", x"AA") completed. ITR : Set interrupts
```

All procedures with:

- Positive acknowledge If wanted
- Alert message and mismatch report
- Alert count and ctrl

Lot's of free UVVM BFMs and VVCs

- AXI4-lite
- AXI4 Full
- AXI-Stream Transmit and Receive
- UART Transmit and Receive
- SBI
- SPI Transmit and Receive
- I2C Transmit and Receive
- GPIO
- Avalon MM
- Avalon Stream Transmit and Receive
- RGMII Transmit and Receive
- GMII Transmit and Receive
- Ethernet Transmit and Receive
- Wishbone
- Clock Generator
- Error Injector

All:

- Free
- Open Source
- Well documented
- Example Testbenches

The largest collection of VHDL Interface Models

VVC: VHDL Verif. Comps.
- Includes the corresponding BFM
Allows:
- Simultaneous interface handling
- Synchronization of interfaces
- Skewing between interfaces
- Additional protocol checkers
- Local sequencers
- Activity detection
- Simple reuse between projects

Specification Coverage

- Assure that all requirements in a specification have been verified
- Also known as Requirements coverage (or Req. tracking / traceability)

- Several similarities with functional coverage, but
  - Most requirements cannot be marked as verified or covered just from one or a few values.
  - Most often involves multiple steps in a test sequence
    - With checks of different values and responses during and after the sequence
  - Application area and customers often require different reporting views

Tracing the Requirements

- (Assure that all requirements in a specification have been verified)

1. Specify all requirements

Requirement Label	Description
MOTOR_R1	The acceleration shall be ***
MOTOR_R2	The top speed shall be given by ***
MOTOR_R3	The deceleration shall be ***
MOTOR_R4	The final position shall be ***

2. Report coverage from test sequencer(s) (or other TB parts)
3. Generate summary report
   - Coverage per requirement
   - Test cases covering each requirement
   - Requirements covered by each Test case
   - Accumulate over multiple Test cases

Requirements Traceability Matrix

Mandatory for Safety and Mission Critical (Strictly required by ESA)
Strongly recommended for good quality assurance
Expensive tools exist…

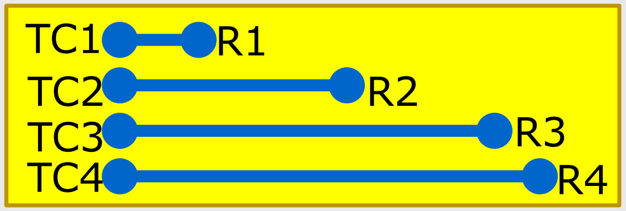
Efficient Specification Coverage

- Free solutions exist to report that a testcase finished successfully
  - BUT - reporting that a testcase has finished
    and accumulating finished testcases - is not sufficient

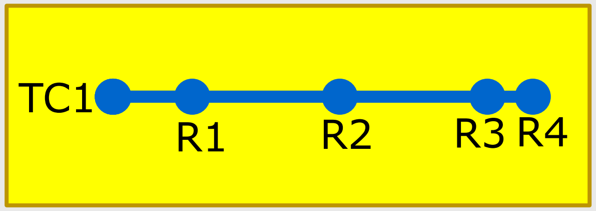
- What if multiple requirements are covered by the same testcase?

Requirement Label	Description
MOTOR_R1	The acceleration shall be ***
MOTOR_R2	The top speed shall be given by ***
MOTOR_R3	The deceleration shall be ***
MOTOR_R4	The final position shall be ***

  - E.g. Moving/turning something to a to a given position
    R1: Acceleration R2: Speed R3: Deceleration 4: Position etc..



VS



- Generates various types of reports

Requirements Traceability Matrix

Example case

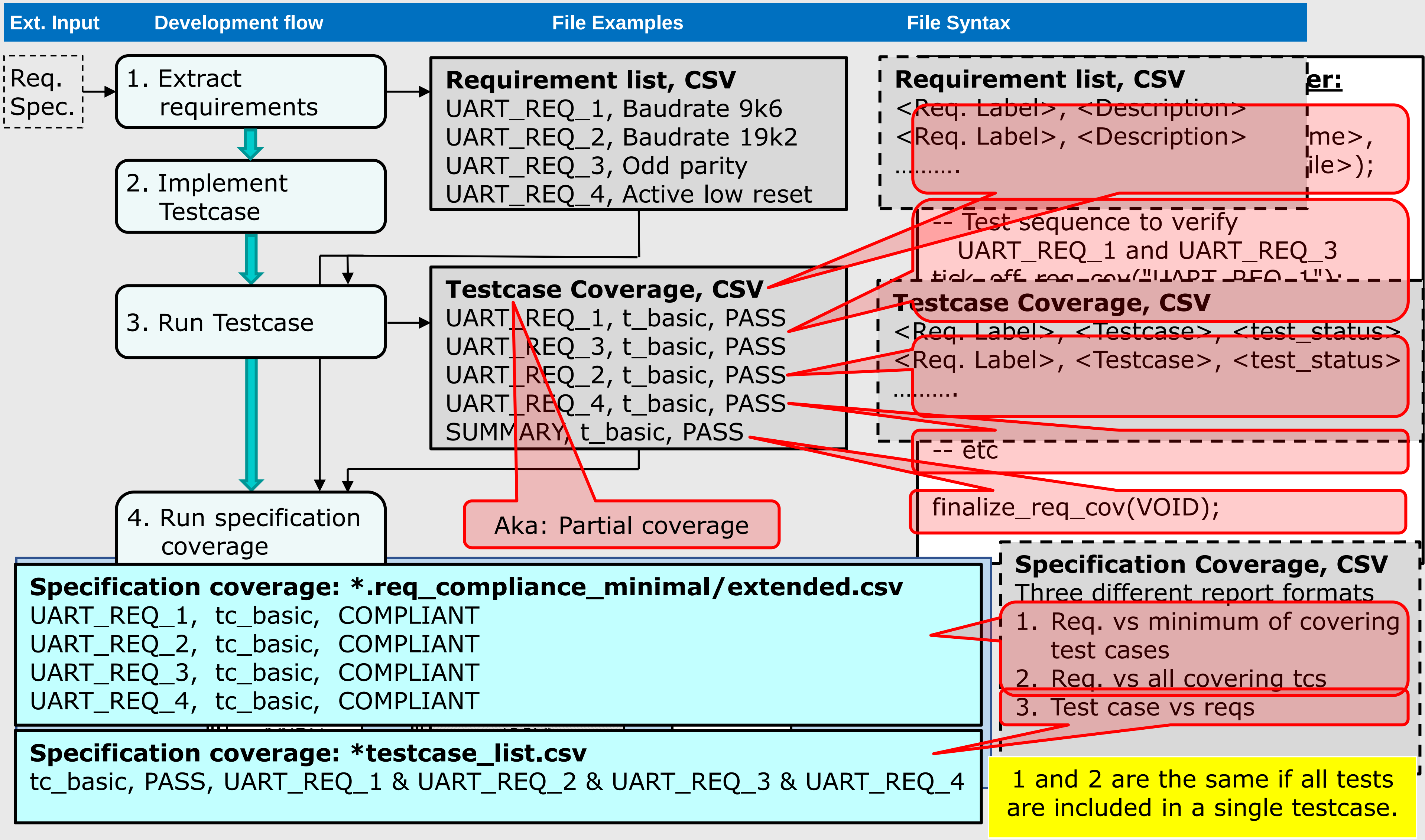
- UART
  - Showing only 4 requirements for simplicity

Requirement Label	**Description**
UART_REQ_1	The device UART interface shall accept a baud rate of 9600kbps.
UART_REQ_2	The device UART interface shall accept a baud rate of 19k2 bps.
UART_REQ_3	The device UART interface shall accept an odd parity
UART_REQ_4	The device reset shall be active low.

- Starting with a **single test case testing all requirements**

Introduction & Simple Case

Simplified overview



Requirement list, CSV
UART_REQ_1, Baudrate 9k6
UART_REQ_2, Baudrate 19k2
UART_REQ_3, Odd parity
UART_REQ_4, Active low reset

Testcase Coverage, CSV
UART_REQ_1, t_basic, PASS
UART_REQ_3, t_basic, PASS
UART_REQ_2, t_basic, PASS
UART_REQ_4, t_basic, PASS
SUMMARY, t_basic, PASS

Requirement list, CSV
<Req. Label>, <Description>
<Req. Label>, <Description>
……….

er:
me>,
ile>);

-- Test sequence to verify
UART_REQ_1 and UART_REQ_3
tick_off_req_cov("UART_REQ_1");

Testcase Coverage, CSV
<Req. Label>, <Testcase>, <test_status>
<Req. Label>, <Testcase>, <test_status>
……….

-- etc

finalize_req_cov(VOID);

Specification Coverage, CSV
Three different report formats
1. Req. vs minimum of covering test cases
2. Req. vs all covering tcs
3. Test case vs reqs

Specification coverage: *.req_compliance_minimal/extended.csv
UART_REQ_1, tc_basic, COMPLIANT
UART_REQ_2, tc_basic, COMPLIANT
UART_REQ_3, tc_basic, COMPLIANT
UART_REQ_4, tc_basic, COMPLIANT

Specification coverage: *testcase_list.csv
tc_basic, PASS, UART_REQ_1 & UART_REQ_2 & UART_REQ_3 & UART_REQ_4

1 and 2 are the same if all tests are included in a single testcase.

Multiple testcases – simple usage

- Normally – Any test can be run in any testcase
- For every single testcase:
  - Inside testcase:
    initialize_req_cov(), N * tick_off_req_cov(), finalize_req_cov()
  - All use the same Requirement list
  - Unique coverage file per test case – **with testcase name included**
- May have multiple testcases inside the same test sequencer

- After running all test cases:
  Run Python script as before, but include all coverage files

Multiple Testcases

- Each test case in VHDL generates a Partial Coverage File (CSV)

Partial coverage
'partial_cov_tc_basic.csv'

```
TESTCASE_NAME: tc_basic
UART_REQ_1, tc_basic,PASS
UART_REQ_3, tc_basic,PASS
SUMMARY, tc_basic, PASS
```

Partial coverage
'partial_cov_tc_19k2.csv'

```
TESTCASE_NAME: tc_19k2

UART_REQ_2, tc_19k2,PASS
UART_REQ_3, tc_19k2,PASS
UART_REQ_4, tc_19k2,PASS

SUMMARY, tc_19k2, PASS
```

Partial coverage
'partial_cov_tc_reset.csv'

```
TESTCASE_NAME: tc_reset

UART_REQ_5,tc_reset,PASS

SUMMARY, tc_reset, PASS
```

Not necessarily the best split into testcases for the UART, but illustrates the usage.

***.req_compliance_minimal.csv**

```
UART_REQ_1, tc_basic, COMPLIANT
UART_REQ_2, tc_19k2,  COMPLIANT
UART_REQ_3, tc_19k2,  COMPLIANT
UART_REQ_4, tc_19k2,  COMPLIANT
UART_REQ_5, tc_reset, COMPLIANT
```

***. req_compliance_extended.csv**

```
UART_REQ_1, tc_basic,  COMPLIANT
UART_REQ_2, tc_19k2,  COMPLIANT
UART_REQ_3, tc_basic & tc_19k2, COMPLIANT
UART_REQ_4, tc_19k2,  COMPLIANT
UART_REQ_5, tc_reset,  COMPLIANT
```

Specification Coverage
Three different report format
1. Req. vs minimum of covering test cases
2. Req. vs all covering tcs
3. Test case vs reqs

***.testcase_list.csv**

```
tc_basic, PASS, UART_REQ_1 & UART_REQ_3
tc_19k2, PASS, UART_REQ_2 & UART_REQ_3 & UART_REQ_4
tc_reset, PASS, UART_REQ_5
```

What if a test fails?

All pass

One or more fail

Partial coverage files

```
t_19k2_cov  (from t_19k2)
UART_REQ_2, t_19k2, PASS
UART_REQ_3, t_19k2, PASS
UART_REQ_4, t_19k2, PASS
SUMMARY, t_19k2, PASS
```

```
t_19k2_cov  (from t_19k2)
UART_REQ_2, t_19k2, PASS
UART_REQ_3, t_19k2, PASS
UART_REQ_4, t_19k2, FAIL
SUMMARY, t_19k2, FAIL
```

Caused by error in testcase

Coverage summary files

```
Specification coverage, CSV  (1)
UART_REQ_1, t_basic, COMPLIANT
UART_REQ_2, t_19k2, COMPLIANT
UART_REQ_3, t_basic, COMPLIANT
UART_REQ_4, t_19k2, COMPLIANT
```

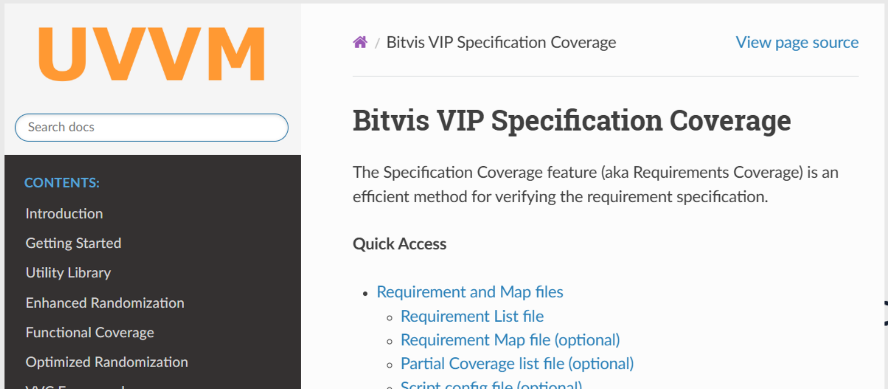
```
Specification coverage, CSV  (1)
UART_REQ_1, t_basic, COMPLIANT
UART_REQ_2, check *.req_non_compliance.csv, NON_COMPLIANT
UART_REQ_3, check *.req_non_compliance.csv, NON_COMPLIANT
UART_REQ_4, check *.req_non_compliance.csv, NON_COMPLIANT
```

UART_REQ_2/3: Caused by failing testcase

UART_REQ_4: Because the requirement failed

Advanced spec. cov.

- May specify required testcase for any given requirement
- May specify that at least one of multiple testcases must pass
  (If the others have not been executed. Cannot have any failed)
- May specify that a requirement is tested in multiple testcases
- May map requirements in one file to requirements in another
  - Thus allowing reusable TBs with coverage included
  - Also allows compound requirement to be split into multiple more detailed requirements
- … and more features.
  See uvvm.github.io



Bitvis VIP Specification Coverage

The Specification Coverage feature (aka Requirements Coverage) is an efficient method for verifying the requirement specification.

Quick Access

- Requirement and Map files
  - Requirement List file
  - Requirement Map file (optional)
  - Partial Coverage list file (optional)
  - Script config file (optional)

Getting started with Spec. Cov
- A step-by-step demo

- Start by including UVVM utility library and Spec.Cov

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

library uvvm_util;
context uvvm_util.uvvm_util_context;

library bitvis_vip_spec_cov;
use bitvis_vip_spec_cov.spec_cov_pkg.all;
```

- Then we are ready to go…
  - To write commands inside our testbench

Simple case – with no test cases specified

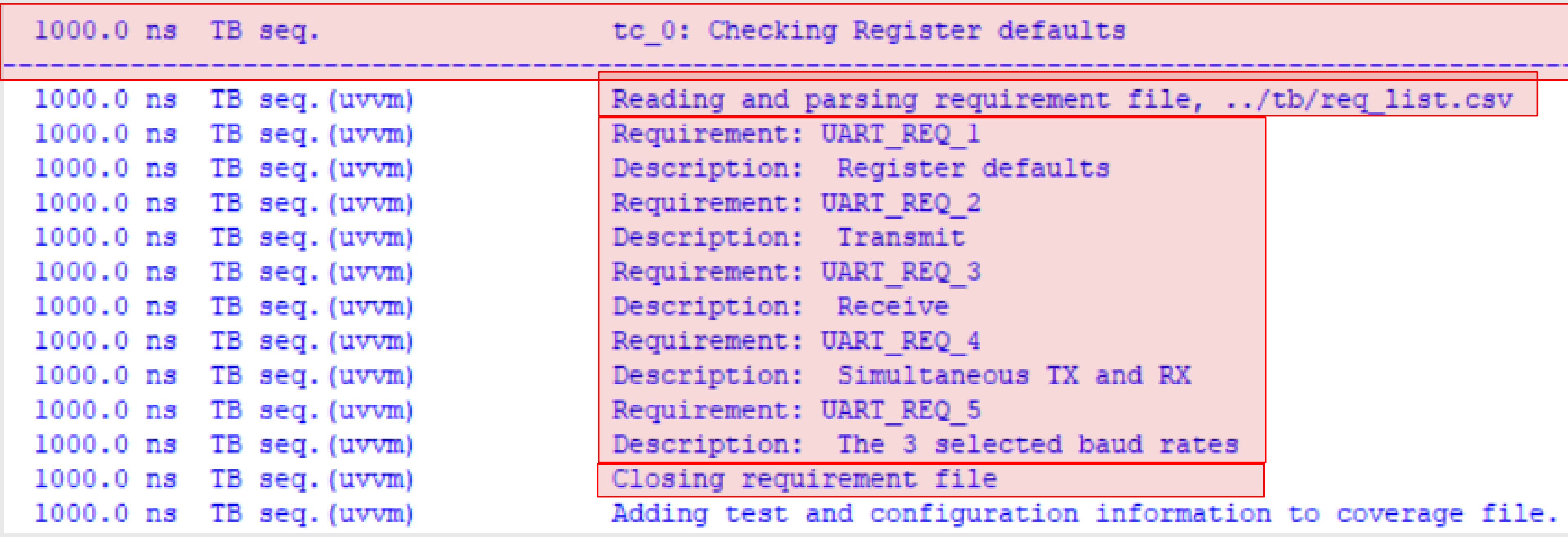
Requirements list: 'req_list.csv'

```
UART_REQ_1, Register defaults
UART_REQ_2, Transmit
UART_REQ_3, Receive
UART_REQ_4, Simultaneous TX & RX
UART_REQ_5, The 3 selected baud rates
```

- **Two parameters only**
- **Testcase not specified**

- **Single testcase only**
- **All Reqs. Covered**

```
log(ID_LOG_HDR, v_testcase & ": Checking Register defaults");
------------------------------------------------------------
initialize_req_cov(v_testcase, C_REQ_LIST_FILE, C_PARTIAL_COV_FILE);
```

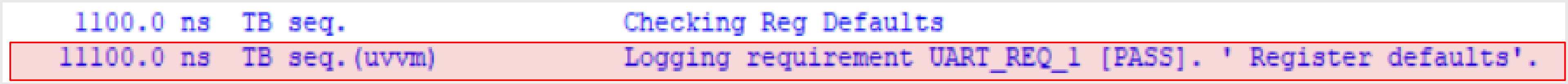


```
1000.0 ns  TB seq.               tc_0: Checking Register defaults
-------------------------------------------------------------------------------------------
1000.0 ns  TB seq.(uvvm)         Reading and parsing requirement file, ../tb/req_list.csv
1000.0 ns  TB seq.(uvvm)         Requirement: UART_REQ_1
1000.0 ns  TB seq.(uvvm)         Description:  Register defaults
1000.0 ns  TB seq.(uvvm)         Requirement: UART_REQ_2
1000.0 ns  TB seq.(uvvm)         Description:  Transmit
1000.0 ns  TB seq.(uvvm)         Requirement: UART_REQ_3
1000.0 ns  TB seq.(uvvm)         Description:  Receive
1000.0 ns  TB seq.(uvvm)         Requirement: UART_REQ_4
1000.0 ns  TB seq.(uvvm)         Description:  Simultaneous TX and RX
1000.0 ns  TB seq.(uvvm)         Requirement: UART_REQ_5
1000.0 ns  TB seq.(uvvm)         Description:  The 3 selected baud rates
1000.0 ns  TB seq.(uvvm)         Closing requirement file
1000.0 ns  TB seq.(uvvm)         Adding test and configuration information to coverage file.
```

Simple case: Coverage (1)

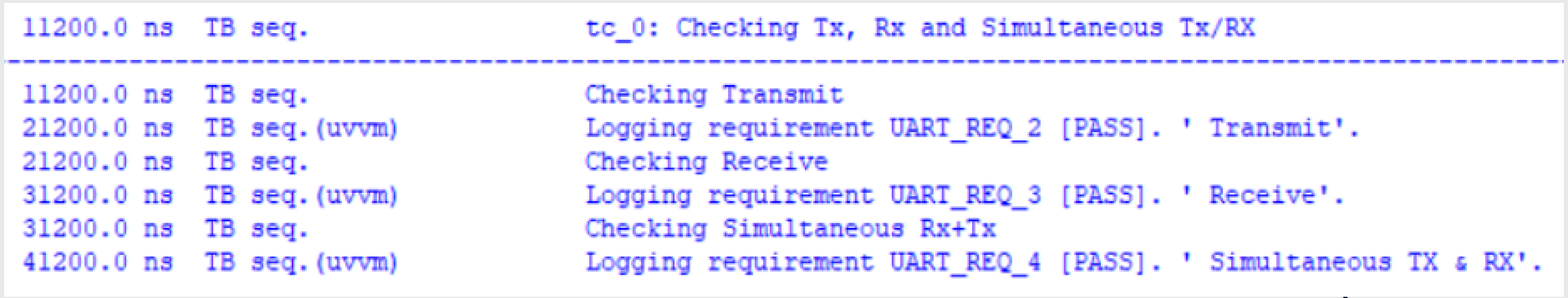
```
UART_REQ_1, Register defaults
UART_REQ_2, Transmit
UART_REQ_3, Receive
UART_REQ_4, Simultaneous TX & RX
UART_REQ_5, The 3 selected baud rates
```

```
log("Checking Reg Defaults");
***** Do all relevant checks;
tick_off_req_cov("UART_REQ_1");
```



```
   1100.0 ns  TB seq.                Checking Reg Defaults
  11100.0 ns  TB seq.(uvvm)          Logging requirement UART_REQ_1 [PASS]. ' Register defaults'.
```

```
log(ID_LOG_HDR, tc_0: Checking Tx, Rx and Simultaneous Tx/RX");
log("Checking Transmit");
  *******
tick_off_req_cov("UART_REQ_2");
log("Checking Receive");
   *******
tick_off_req_cov("UART_REQ_3")
log("Checking Simultaneous Rx+Tx");
   *******
tick_off_req_cov("UART_REQ_4");
```

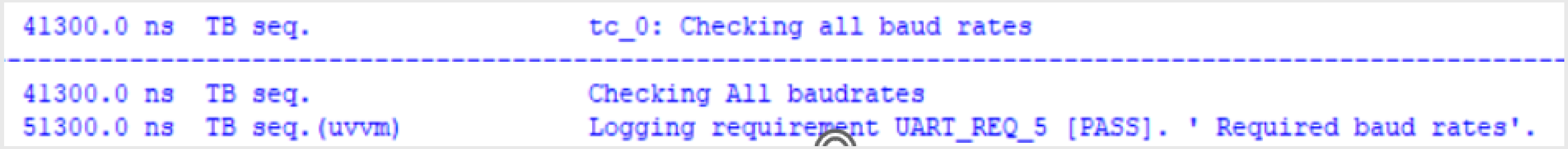


```
11200.0 ns  TB seq.                tc_0: Checking Tx, Rx and Simultaneous Tx/RX
----------------------------------------------------------------------------------------------------
11200.0 ns  TB seq.                Checking Transmit
21200.0 ns  TB seq.(uvvm)          Logging requirement UART_REQ_2 [PASS]. ' Transmit'.
21200.0 ns  TB seq.                Checking Receive
31200.0 ns  TB seq.(uvvm)          Logging requirement UART_REQ_3 [PASS]. ' Receive'.
31200.0 ns  TB seq.                Checking Simultaneous Rx+Tx
41200.0 ns  TB seq.(uvvm)          Logging requirement UART_REQ_4 [PASS]. ' Simultaneous TX & RX'.
```

Simple case: Coverage (2)

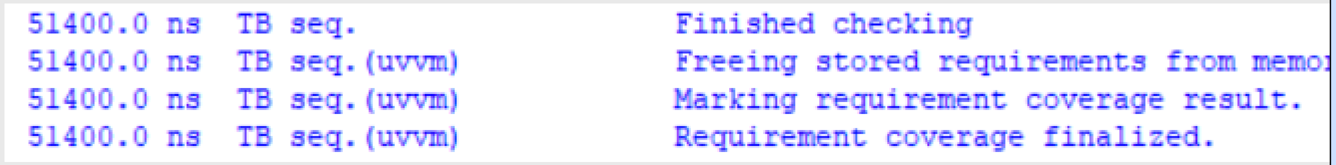
```
UART_REQ_1, Register defaults
UART_REQ_2, Transmit
UART_REQ_3, Receive
UART_REQ_4, Simultaneous TX & RX
UART_REQ_5, The 3 selected baud rates
```

```
log("Checking All baudrates");
  *******
tick_off_req_cov("UART_REQ_5");
```



```
41300.0 ns  TB seq.               tc_0: Checking all baud rates
-----------------------------------------------------------------------------------------------------
41300.0 ns  TB seq.               Checking All baudrates
51300.0 ns  TB seq.(uvvm)         Logging requirement UART_REQ_5 [PASS]. ' Required baud rates'.
```

```
log("Finished checking");
finalize_req_cov(VOID);
```



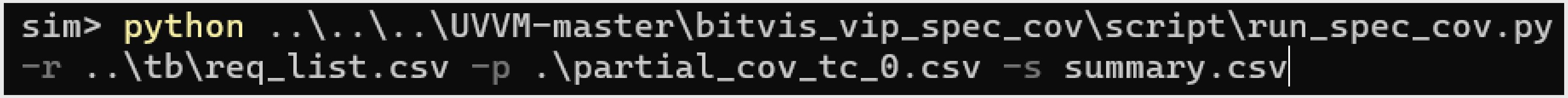
```
51400.0 ns  TB seq.             Finished checking
51400.0 ns  TB seq.(uvvm)       Freeing stored requirements from memo
51400.0 ns  TB seq.(uvvm)       Marking requirement coverage result.
51400.0 ns  TB seq.(uvvm)       Requirement coverage finalized.
```

Partial coverage file (simplified)

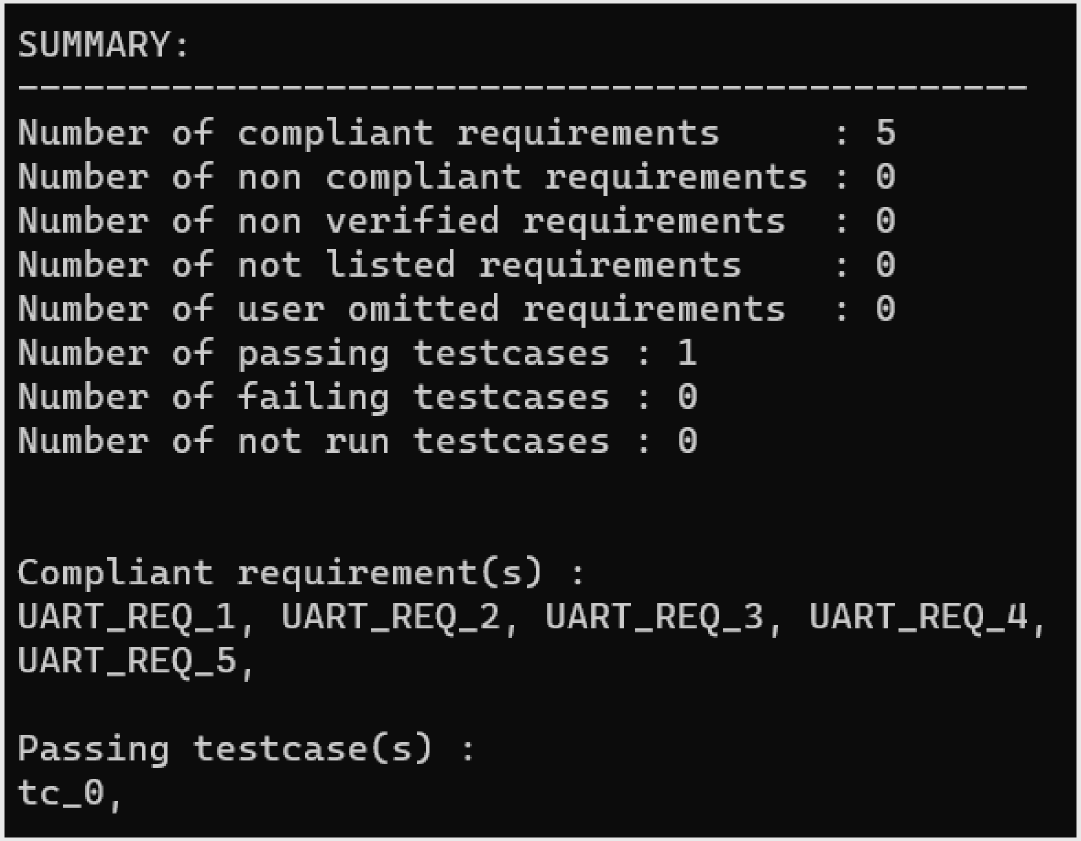
```
TESTCASE_NAME: tc_0

UART_REQ_1,tc_0,PASS
UART_REQ_2,tc_0,PASS
UART_REQ_3,tc_0,PASS
UART_REQ_4,tc_0,PASS
UART_REQ_5,tc_0,PASS
SUMMARY, tc_0, PASS
```

Simple case: Coverage summary

```
UART_REQ_1, Register defaults
UART_REQ_2, Transmit
UART_REQ_3, Receive
UART_REQ_4, Simultaneous TX & RX
UART_REQ_5, Required baud rates
```



```
sim> python ..\..\..\UVVM-master\bitvis_vip_spec_cov\script\run_spec_cov.py
-r ..\tb\req_list.csv -p .\partial_cov_tc_0.csv -s summary.csv
```



```
SUMMARY:
----------------------------------------------
Number of compliant requirements      : 5
Number of non compliant requirements : 0
Number of non verified requirements  : 0
Number of not listed requirements    : 0
Number of user omitted requirements  : 0
Number of passing testcases : 1
Number of failing testcases : 0
Number of not run testcases : 0


Compliant requirement(s) :
UART_REQ_1, UART_REQ_2, UART_REQ_3, UART_REQ_4,
UART_REQ_5,

Passing testcase(s) :
tc_0,
```

Partial coverage file

```
TESTCASE_NAME: tc_0

UART_REQ_1,tc_0,PASS
UART_REQ_2,tc_0,PASS
UART_REQ_3,tc_0,PASS
UART_REQ_4,tc_0,PASS
UART_REQ_5,tc_0,PASS
SUMMARY, tc_0, PASS
```

***.req_compliance_minimal/ extended.csv**

```
Requirement,Testcase,Compliance
UART_REQ_1,tc_0,COMPLIANT
UART_REQ_2,tc_0,COMPLIANT
UART_REQ_3,tc_0,COMPLIANT
UART_REQ_4,tc_0,COMPLIANT
UART_REQ_5,tc_0,COMPLIANT
```

Simple case (using requirements): But now with testcase specified

Requirements list: 'req_list.csv'

```
UART_REQ_1, Register defaults, tc_1
UART_REQ_2, Transmit, tc_2
UART_REQ_3, Receive, tc_2
UART_REQ_4, Simultaneous TX and RX, tc_2
UART_REQ_5, The 3 selected baudrates, tc_3
```

- Same as before, but obviously:
  - Will need to run all 3 testcases
  - And Generate partial coverage from all

Partial coverage
'partial_cov_tc1.csv'

```
TESTCASE_NAME: tc_1

UART_REQ_1,tc_1,PASS
SUMMARY, tc_1, PASS
```

Partial coverage
'partial_cov_tc2.csv'

```
TESTCASE_NAME: tc_2

UART_REQ_2,tc_2,PASS
UART_REQ_3,tc_2,PASS
UART_REQ_4,tc_2,PASS
SUMMARY, tc_2, PASS
```

Partial coverage
'partial_cov_tc3.csv'

```
TESTCASE_NAME: tc_3

UART_REQ_5,tc_3,PASS
SUMMARY, tc_3, PASS
```

Preparing Post-processing

- When multiple partial coverage files
  → List them in a text file

'partial_cov_files.txt'

```
../sim/partial_cov_tc_1.csv
../sim/partial_cov_tc_2.csv
../sim/partial_cov_tc_3.csv
```

- Rather than including all arguments on command line
  → Make a config file

'config.txt'

```
-r ../tb/req_list.csv
-p ../tb/partial_cov_files.txt
-s summary.csv
```

- Run Pythion script to generate summary files

***.req_compliance_minimal.csv**

```
UART_REQ_1, tc_1, COMPLIANT
UART_REQ_2, tc_2, COMPLIANT
UART_REQ_3, tc_2, COMPLIANT
UART_REQ_4, tc_2, COMPLIANT
UART_REQ_5, tc_3, COMPLIANT
```

EmLogic

Specification coverage: Summary

- Does not in itself say anything about absolute quality
  - Depends on developer really testing the right stuff
- Depends on a good req. spec.
  - All relevant reqs. included
  - Compound reqs. should be split into testable sub-reqs.
- Given good testcases that do what they are intended to do…:
  - **Specifiction coverage will track and assure that:**
    All requirements in a specification have been verified

It is common to just - sometime during development
- tick off somewhere – that a particular requirement is tested;
often just as a mental exercise..

It is always better to use a written, repeatable and automated approach.

This VIP significantly simplifies such an approach.

UVVM – Spec. Cov. - Summary

- UVVM is used by many FPGA/ASIC designers for:
  - ESA/NASA mission critical
  - DO-254
  - High quality in general
- UVVM specification coverage
  - Was developed in cooperation with ESA (Project 2)
  - Is updated/extended in our current 3rd ESA project
- You may pick any part UVVM without a "lock-in"
  - Works with any other VHDL testbench methodology or legacy TBs
- Will result in

  → Significant quality improvement

  → Significant efficiency improvement

  → Far less boring, time-consuming manual documentation

EmLogic

Get the right FPGA quality
through
Efficient Verification
and Requirements Tracking

Thanks for your attention

et@emlogic.no