EmLogic

UVVM
UVM for VHDL designers – An introduction

DVClub, 28 June 2022

EmLogic

- Independent Design Centre for Embedded Systems and FPGA
- Established 1st of January 2021. **Extreme ramp up**
  - January 2021: 1 person
  - June 2022: → 23 designers (SW:8, HW:3, FPGA:10) - **And still growing…**
- 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

- Very structured infrastructure and 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

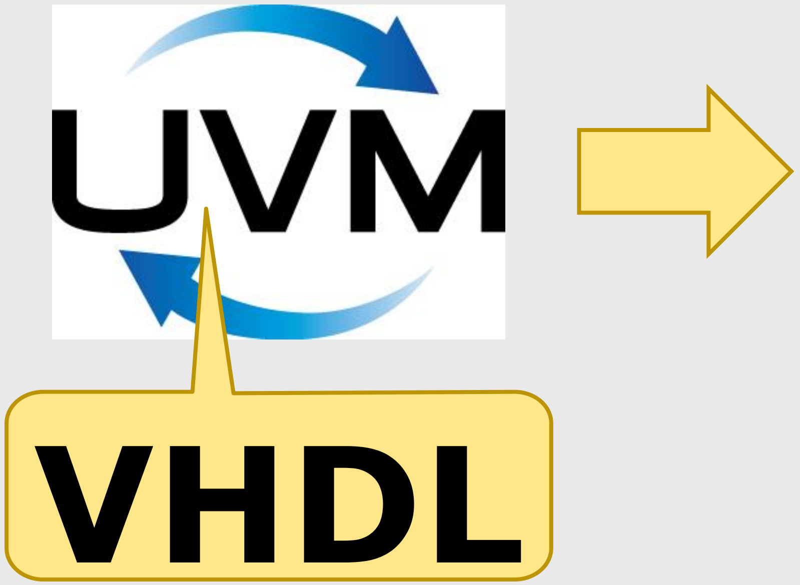
- Included with various simulators
- Runs on GHDL

UVVM: UVM for VHDL designers

VHDL was declared as "dying" already in 2003
- and in 2007, - and ....

BUT - According to Wilson research September 2020:

VHDL for FPGA: >50% world-wide

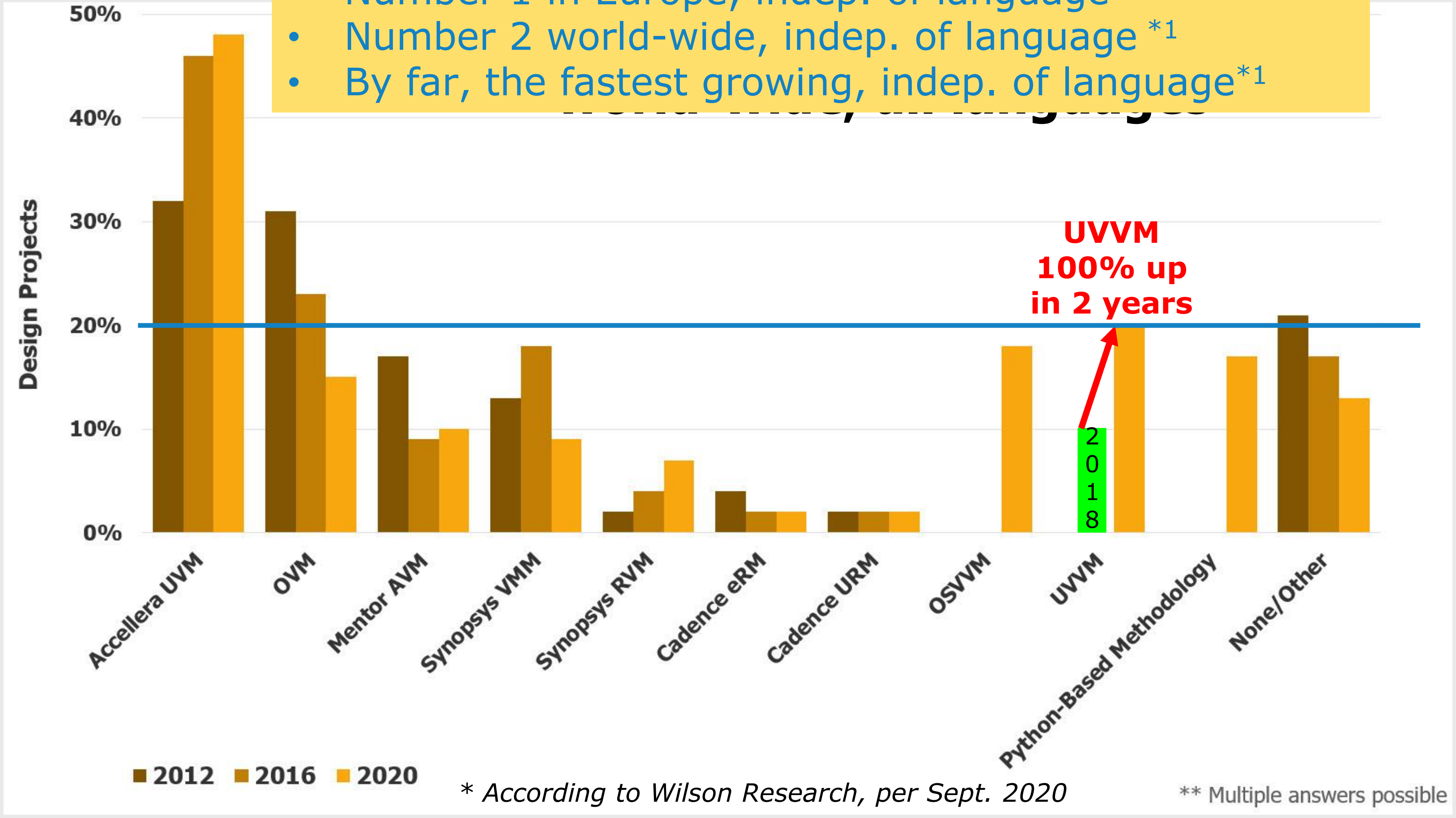


UVVM

- For VHDL designers
- Low user threshold
- Logical evolution on VHDL
- Low cost solution
- More efficient VHDL designers

UVVM – World-wide

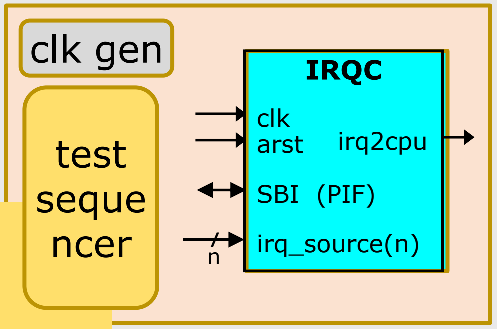
- Number 1 world-wide for VHDL verification *1
- Number 1 in Europe, indep. of language *1
- Number 2 world-wide, indep. of language *1
- By far, the fastest growing, indep. of language*1



* *According to Wilson Research, per Sept. 2020*

Example on test sequencer code and transcript/log

Testbench



```
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, '1', C_CLK_PERIOD, "Set IRQ source 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");
```

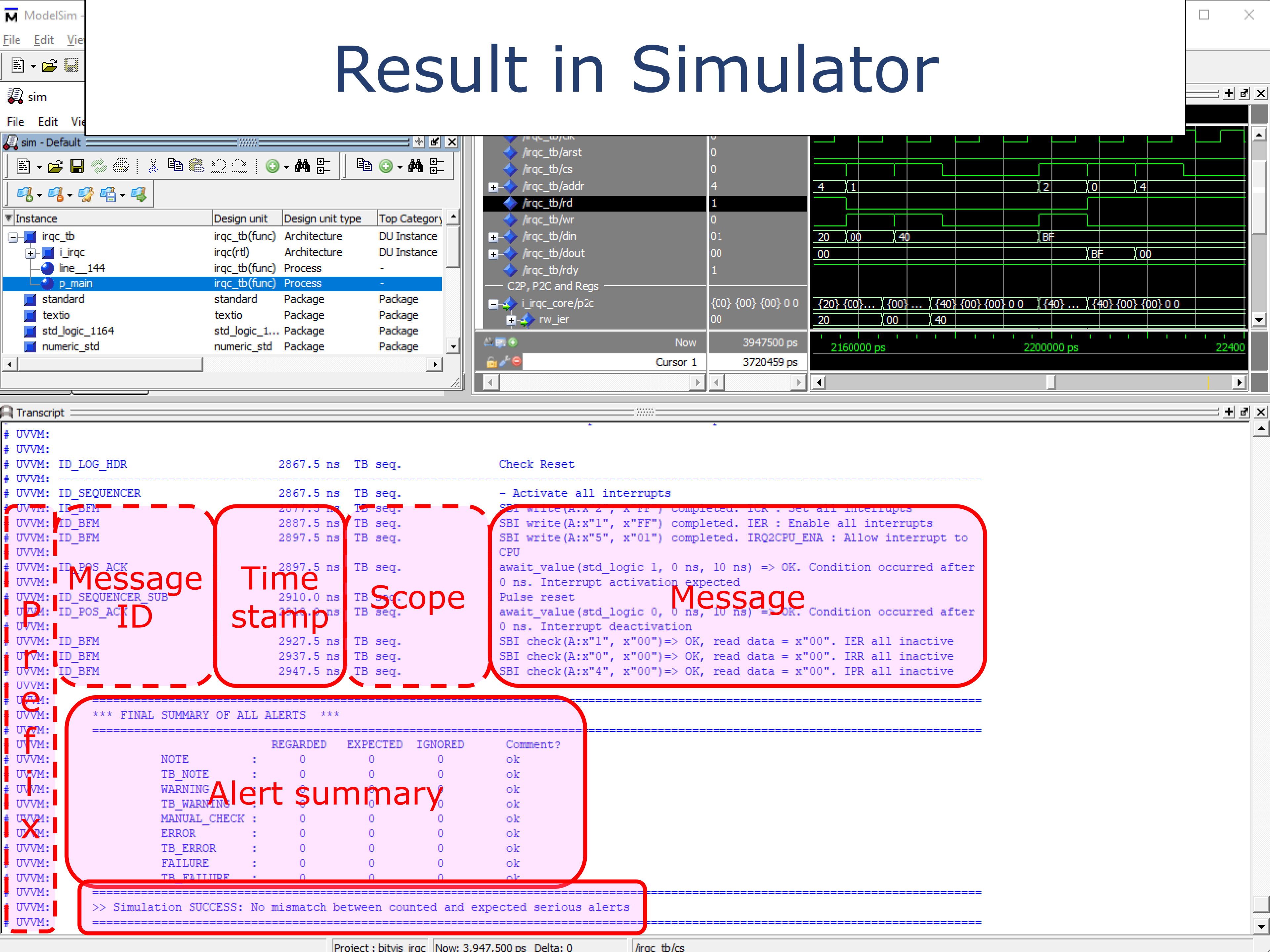
```
2000.0 ns    Check Interrupt trigger clear mechanism
--------------------------------------------------------------------
 110.0 ns    check_value() => OK, for std_logic '0'. irq2cpu default
 727.5 ns    check_stable() => OK. Stable at 0. Stable irq2cpu
1060.0 ns    Pulsed to '1'. Set IRQ source 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

EmLogic

Result in Simulator



```
# UVVM:
# UVVM:
# UVVM:  ID_LOG_HDR                     2867.5 ns  TB seq.            Check Reset
# UVVM:  ----------------------------------------------------------------------------------------------------------------
# UVVM:  ID_SEQUENCER                   2867.5 ns  TB seq.            - Activate all interrupts
# UVVM:  ID_BFM                         2877.5 ns  TB seq.            SBI write(A:x"2", x"FF") completed. ICR : Set all interrupts
# UVVM:  ID_BFM                         2887.5 ns  TB seq.            SBI write(A:x"1", x"FF") completed. IER : Enable all interrupts
# UVVM:  ID_BFM                         2897.5 ns  TB seq.            SBI write(A:x"5", x"01") completed. IRQ2CPU_ENA : Allow interrupt to
# UVVM:                                                               CPU
# UVVM:  ID_POS_ACK                     2897.5 ns  TB seq.            await_value(std_logic 1, 0 ns, 10 ns) => OK. Condition occurred after
# UVVM:                                                               0 ns. Interrupt activation expected
# UVVM:  ID_SEQUENCER_SUB               2910.0 ns  TB seq.            Pulse reset
# UVVM:  ID_POS_ACK                     2910.0 ns  TB seq.            await_value(std_logic 0, 0 ns, 10 ns) => OK. Condition occurred after
# UVVM:                                                               0 ns. Interrupt deactivation
# UVVM:  ID_BFM                         2927.5 ns  TB seq.            SBI check(A:x"1", x"00")=> OK, read data = x"00". IER all inactive
# UVVM:  ID_BFM                         2937.5 ns  TB seq.            SBI check(A:x"0", x"00")=> OK, read data = x"00". IRR all inactive
# UVVM:  ID_BFM                         2947.5 ns  TB seq.            SBI check(A:x"4", x"00")=> OK, read data = x"00". IPR all inactive
# UVVM:
# UVVM:      ================================================================================================================
# UVVM:      *** FINAL SUMMARY OF ALL ALERTS  ***
# UVVM:      ================================================================================================================
# UVVM:                                 REGARDED   EXPECTED  IGNORED      Comment?
# UVVM:                NOTE         :      0          0         0         ok
# UVVM:                TB_NOTE      :      0          0         0         ok
# UVVM:                WARNING      :      0          0         0         ok
# UVVM:                TB_WARNING   :      0          0         0         ok
# UVVM:                MANUAL_CHECK :      0          0         0         ok
# UVVM:                ERROR        :      0          0         0         ok
# UVVM:                TB_ERROR     :      0          0         0         ok
# UVVM:                FAILURE      :      0          0         0         ok
# UVVM:                TB_FAILURE   :      0          0         0         ok
# UVVM:      ================================================================================================================
# UVVM:      >> Simulation SUCCESS: No mismatch between counted and expected serious alerts
# UVVM:      ================================================================================================================
```

```
Instance        Design unit      Design unit type   Top Category
irqc_tb         irqc_tb(func)    Architecture       DU Instance
  i_irqc        irqc(rtl)        Architecture       DU Instance
  line__144     irqc_tb(func)    Process            -
  p_main        irqc_tb(func)    Process            -
standard        standard         Package            Package
textio          textio           Package            Package
std_logic_1164  std_logic_1...   Package            Package
numeric_std     numeric_std      Package            Package
```

```
/irqc_tb/arst    0
/irqc_tb/cs      0
/irqc_tb/addr    4
/irqc_tb/rd      1
/irqc_tb/wr      0
/irqc_tb/din     01
/irqc_tb/dout    00
/irqc_tb/rdy     1
C2P, P2C and Regs
i_irqc_core/p2c  {00} {00} {00} 0 0
rw_ier           00
Now       3947500 ps
Cursor 1  3720459 ps
```

Project : bitvis_irqc   Now: 3,947,500 ps Delta: 0   /irqc_tb/cs

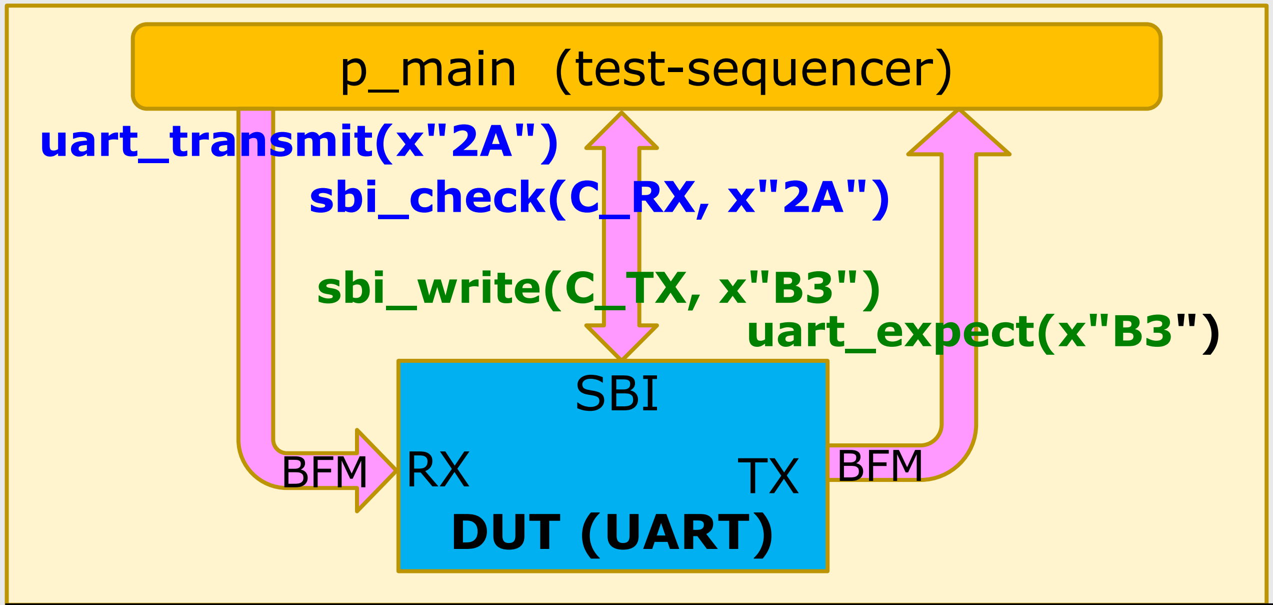
UVVM Utility Library

Testbench infrastructure library

- log(), alert(), report_alert_counters()
- check_value(), await_value()
- check_stable(),   await_stable()
- clock_generator(),   adjustable_clock_generator()
- random(), randomize()
- gen_pulse()
- block_flag(), unblock_flag(), await_unblock_flag()
- await_barrier()
- enable_log_msg(),   disable_log_msg()
- to_string(), fill_string(), to_upper(), replace(), etc…
- normalize_and_check()
- set_log_file_name(),   set_alert_file_name()
- wait_until_given_time_after_rising_edge()
- etc…

EmLogic

Simple data communication



May use Utility Library and provided BFMs

Free, Open source BFMs:

UART, AXI4-lite, SPI, I2C, Avalon MM, AXI4-stream, Avalon stream, GPIO, SBI, GMII, RGMII, …

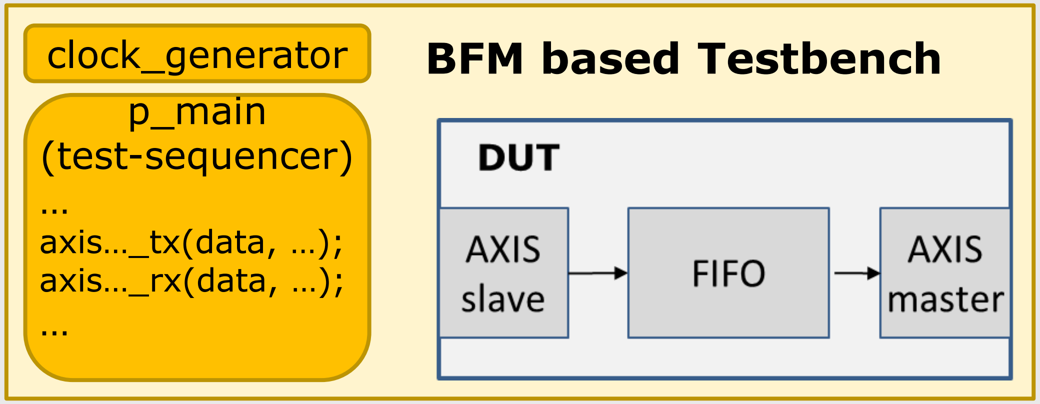
Quick References are provided

```
TB:   172 ns. uart_tb     uart_transmit(x2A) on UART RX
TB:   192 ns. uart_tb     sbi_check(x1, ==> x2A) completed. From UART RX
```

```
TB:   192 ns. uart_tb     sbi_write(x2, ==> xB3) completed. To UART TX
TB: ERROR:
TB:      192 ns. uart_tb
TB:              value was: 'xB2'.  expected 'xB3'.
TB:              (From uart_expect(xB3))
TB:==============================================================
```

EmLogic

AXI-stream - BFM based TB
- as simple as possible



```
…
axis…_tx(data, …);
axis…_rx(data, …);
…
```

- No test harness (for simplicity)
- Sequencer has direct access to DUT signals
  - Thus BFMs from p_main can also see the DUT signals

UVVM_Light (from github)

uvvm_util (library)

```
log, check_value, await_value, etc…
clock_generator()
axistream_transmit(data, ...)      (procedure)
axistream_receive(data, ...)       (procedure)
axistream_expect(data, ...)        (procedure)
etc…
```

- Simplified UVVM
  - For simple usage
- Subset of UVVM
  No VVCs or VCC support
- All BFMs in the same directory and library

Only need to download from Github (clone or zip) and compile (total 5 min)

EmLogic

Resulting transcript +Debug

Note: Removed Prefix and Scope to show on a single line.

```
axistream_transmit(v_byte_array, msg, clk, m_axis);
```

```
ID_BFM                 106.0 ns   axistream_transmit(3B)=> Tx DONE.
```

```
axistream_expect(v_exp_array(0 to 2), "", clk, s_axis);
```

```
ID_BFM                 122.0 ns   axistream_expect(3B)=> OK, received 3B.
```

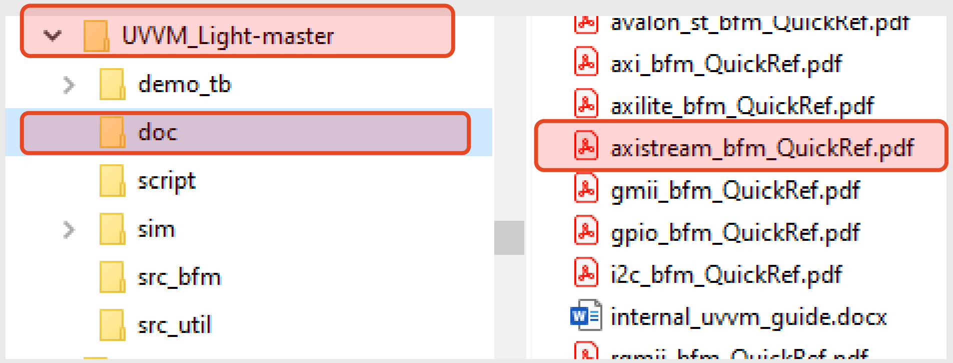
May add more info for debugging

```
enable_log_msg(ID_PACKET_INITIATE);       enable_log_msg(ID_PACKET_DATA);
```

```
ID_PACKET_INITIATE      52.0 ns   axistream_transmit(3B)=>
ID_PACKET_DATA          52.0 ns   axistream_transmit(3B)=> Tx x"00", byte# 0.
ID_PACKET_DATA          68.0 ns   axistream_transmit(3B)=> Tx x"01", byte# 1.
ID_PACKET_DATA          82.0 ns   axistream_transmit(3B)=> Tx x"02", byte# 2.
ID_PACKET_COMPLETE     106.0 ns   axistream_transmit(3B)=> Tx DONE.
```

May add similar debugging info for data reception

Documentation BFM

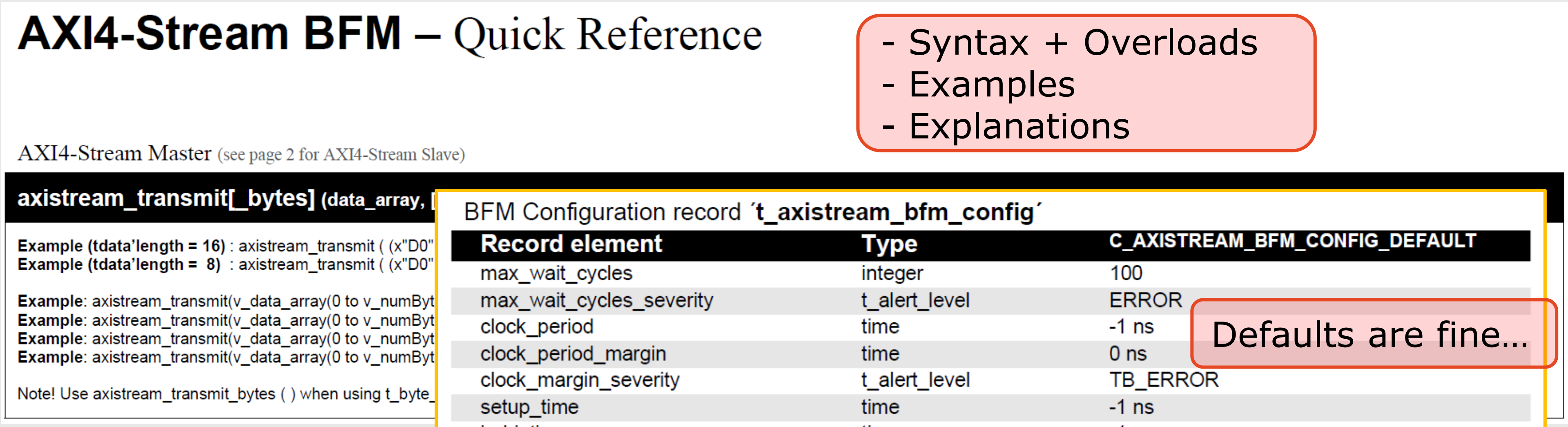


UVVM_Light-master
demo_tb
doc
script
sim
src_bfm
src_util

avalon_st_bfm_QuickRef.pdf
axi_bfm_QuickRef.pdf
axilite_bfm_QuickRef.pdf
axistream_bfm_QuickRef.pdf
gmii_bfm_QuickRef.pdf
gpio_bfm_QuickRef.pdf
i2c_bfm_QuickRef.pdf
internal_uvvm_guide.docx
rgmii_bfm_QuickRef.pdf

Similar docs for all BFMs

Documentation BFM



AXI4-Stream BFM – Quick Reference

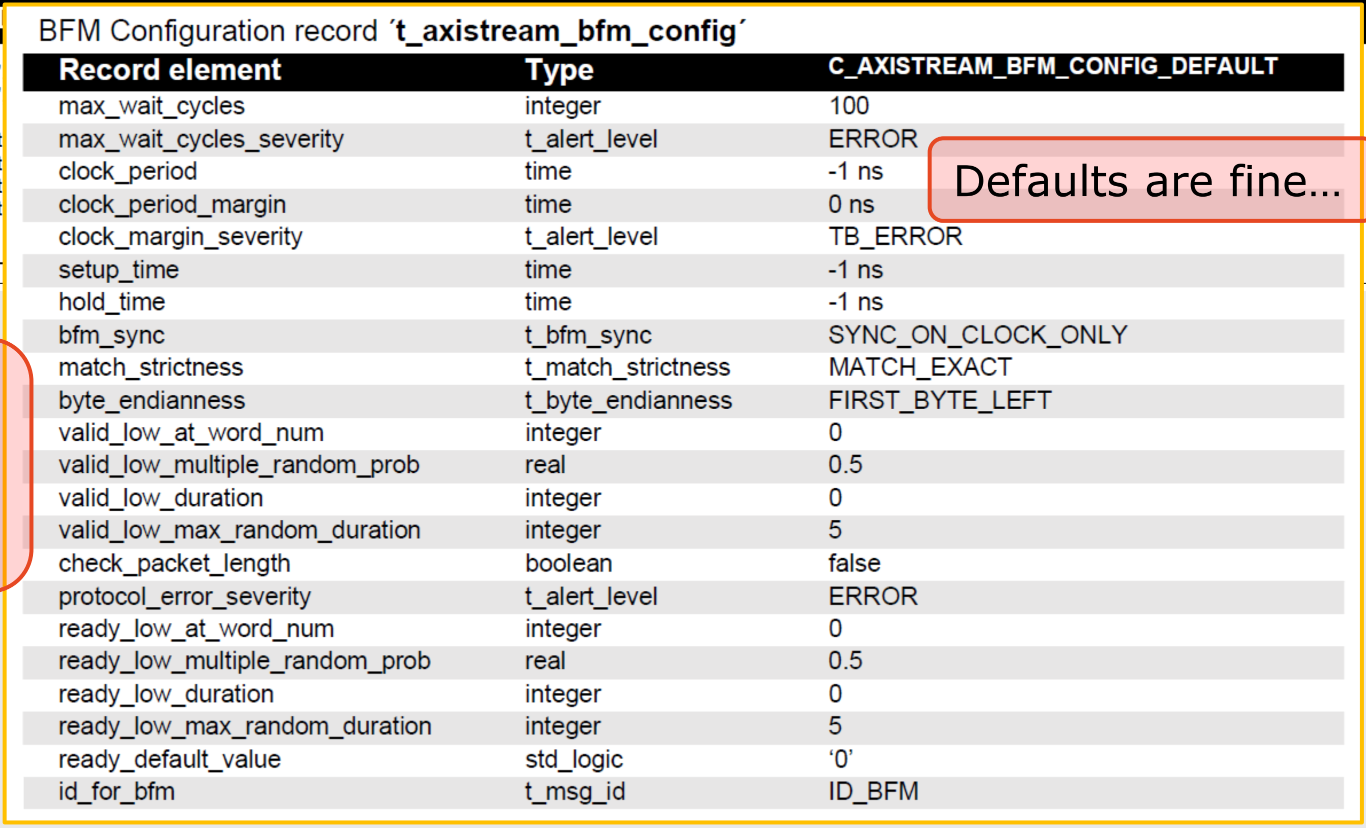
AXI4-Stream Master (see page 2 for AXI4-Stream Slave)

axistream_transmit[_bytes] (data_array, [

Example (tdata'length = 16) : axistream_transmit ((x"D0"
Example (tdata'length = 8) : axistream_transmit ((x"D0"

Example: axistream_transmit(v_data_array(0 to v_numByt
Example: axistream_transmit(v_data_array(0 to v_numByt
Example: axistream_transmit(v_data_array(0 to v_numByt
Example: axistream_transmit(v_data_array(0 to v_numByt

Note! Use axistream_transmit_bytes ( ) when using t_byte_



BFM Configuration record ´**t_axistream_bfm_config**´

Record element	**Type**	**C_AXISTREAM_BFM_CONFIG_DEFAULT**
max_wait_cycles	integer	100
max_wait_cycles_severity	t_alert_level	ERROR
clock_period	time	-1 ns
clock_period_margin	time	0 ns
clock_margin_severity	t_alert_level	TB_ERROR
setup_time	time	-1 ns
hold_time	time	-1 ns
bfm_sync	t_bfm_sync	SYNC_ON_CLOCK_ONLY
match_strictness	t_match_strictness	MATCH_EXACT
byte_endianness	t_byte_endianness	FIRST_BYTE_LEFT
valid_low_at_word_num	integer	0
valid_low_multiple_random_prob	real	0.5
valid_low_duration	integer	0
valid_low_max_random_duration	integer	5
check_packet_length	boolean	false
protocol_error_severity	t_alert_level	ERROR
ready_low_at_word_num	integer	0
ready_low_multiple_random_prob	real	0.5
ready_low_duration	integer	0
ready_low_max_random_duration	integer	5
ready_default_value	std_logic	'0'
id_for_bfm	t_msg_id	ID_BFM

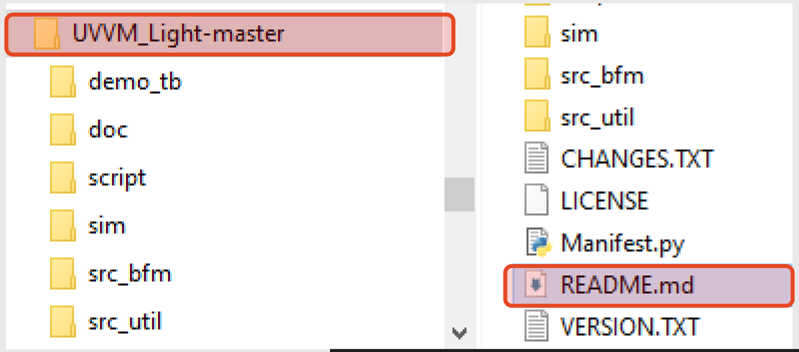
- Syntax + Overloads
- Examples
- Explanations

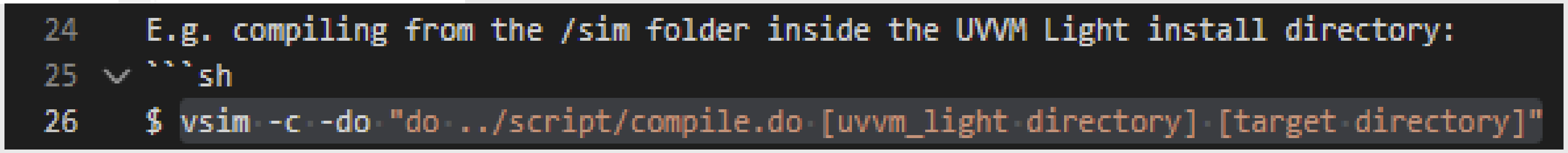
Defaults are fine…

Configuration
- Protocol Behaviour
- Compliance checking
- Simulation set-up

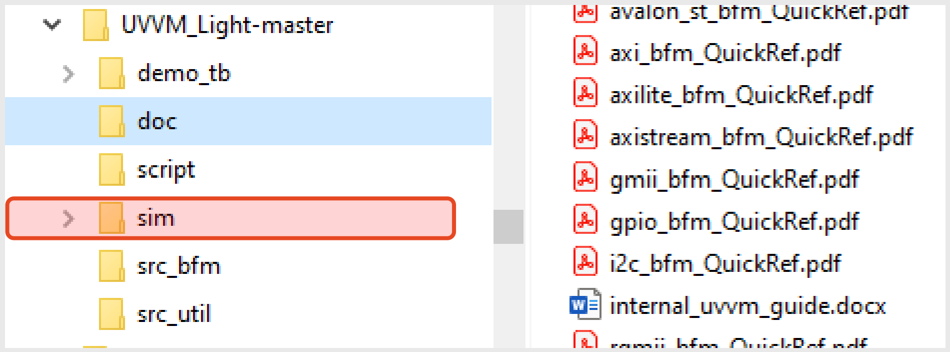
EmLogic

Compiling UVVM Light





~~~
E.g. compiling from the /sim folder inside the UVVM Light install directory:
```sh
$ vsim -c -do "do ../script/compile.do [uvvm_light directory] [target directory]"
~~~



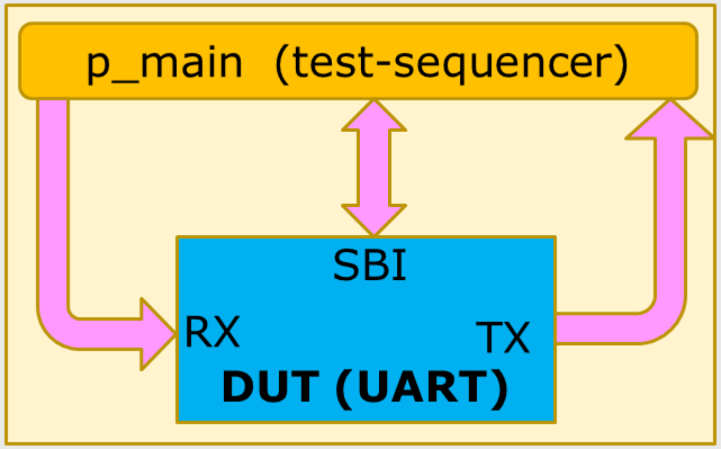
```
vsim -c -do "do ../script/compile.do ../ ."
```

EmLogic

BFM procedures are not sufficient

BFM: Defined here as a procedure only

- BFMs are great for simple testbenches
  - Dedicated procedures in a simple package
  - Just reference and call from a process
- BUT
  - A process can only do one thing at a time
    - Either execute that BFM
    - **Or** execute another BFM
    - **Or** do something else
- To do more than one thing:
  → Need an entity (or component)
  (VC = Verification Component)



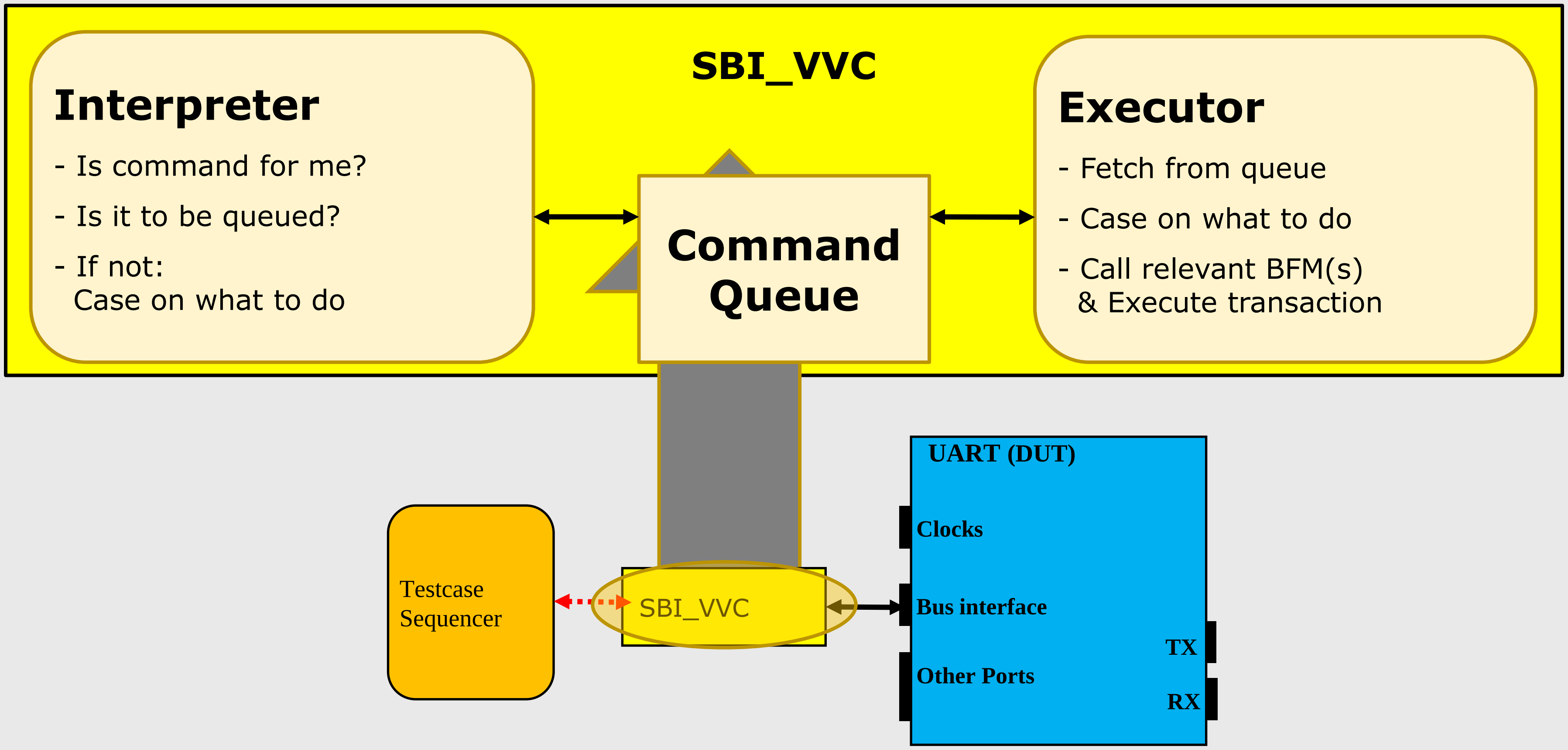
sbi_write(C_TX, x"B3")

uart_expect(x"B3")

VVC: VHDL Verification Component (UVVM VC with extended functionality)

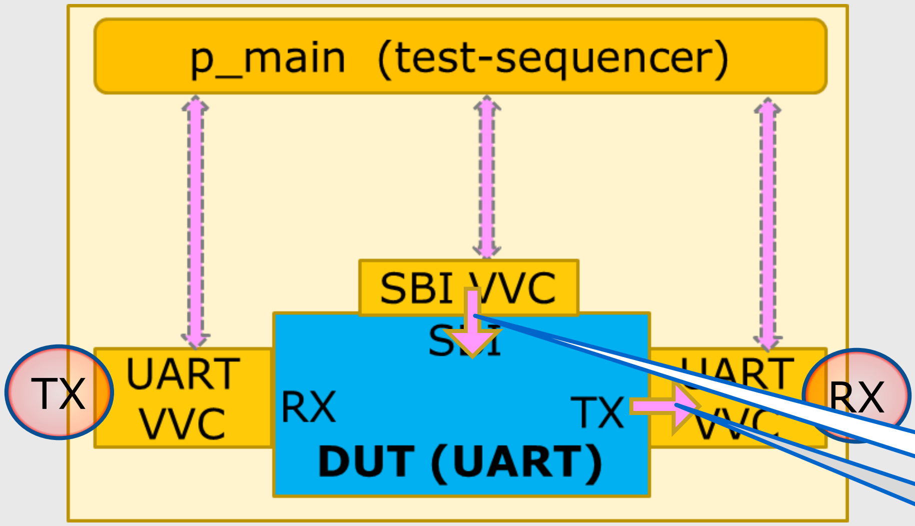
EmLogic

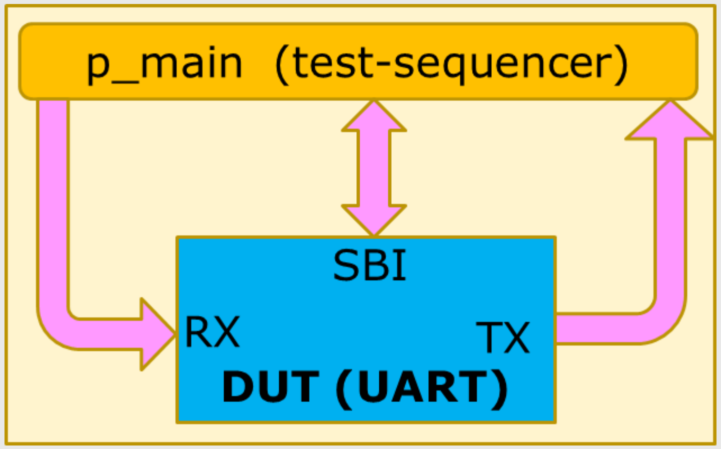
VVC: VHDL Verification Component



EmLogic

BFM to VVC: How?





sbi_write(SBI_VVCT,1, C_TX, x"B3")

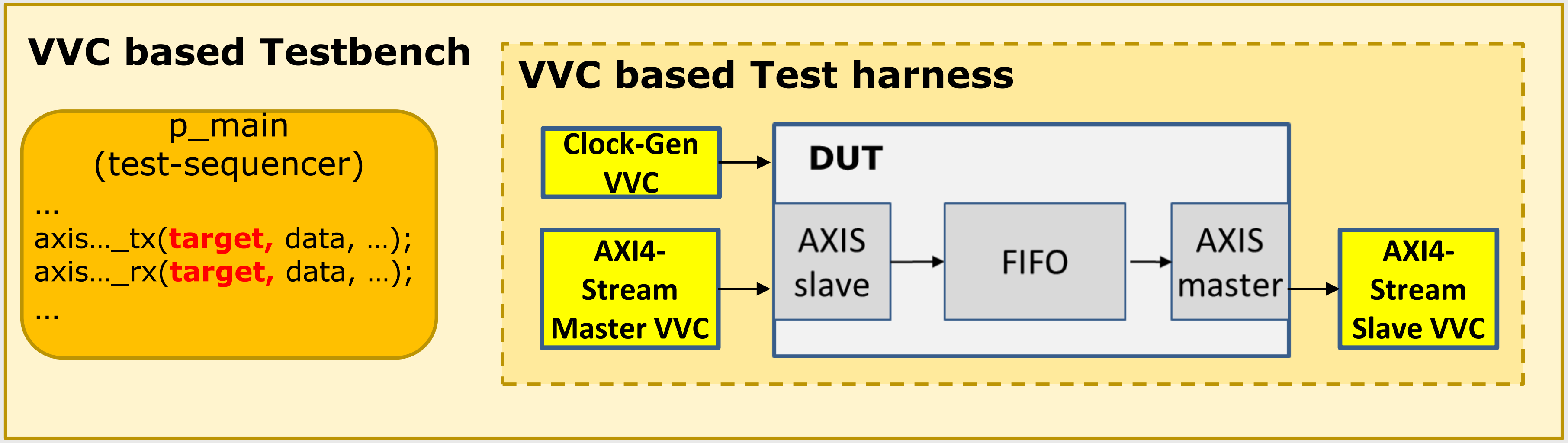
uart_expect(UART_VVCT, 1, RX, x"B3")

sbi_write(C_TX, x"B3")

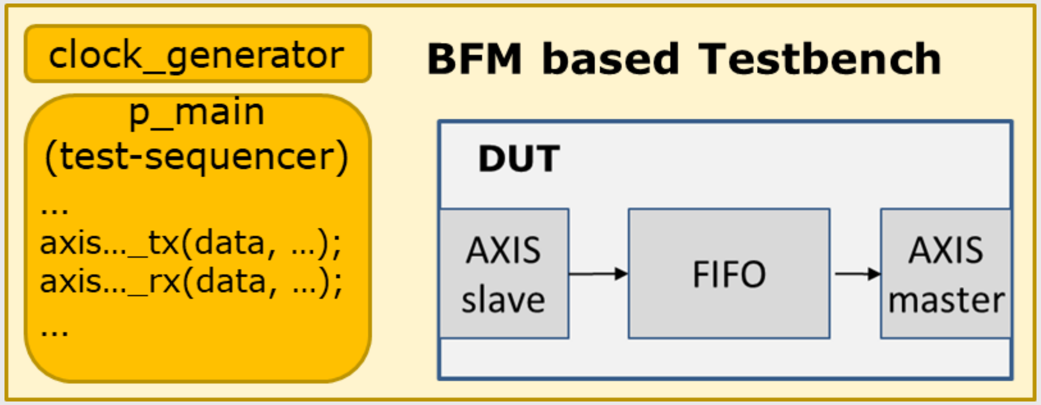
uart_expect(x"B3")

UVVM VVCs also include:
Delay-insertion, command queuing, completion detection, activity registration, multicast & broadcast, termination, set-up, data fetch, multi-channel support, interface checkers, scoreboards, transaction info, local sequencers, etc …

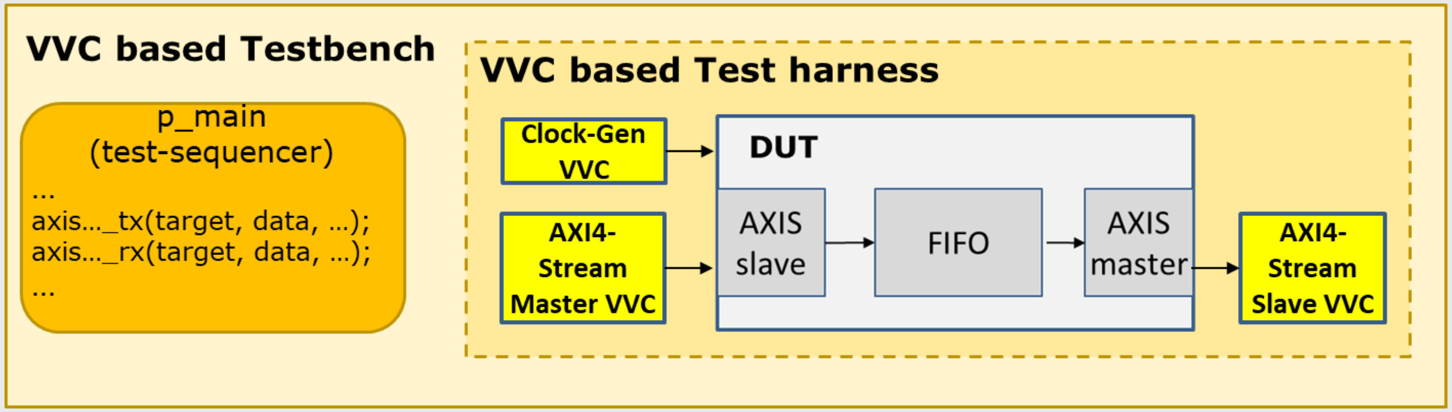
AXI-stream - VVC based TB (1)



axistream_transmit(**target,** data, …);
axistream_expect(**target,** data, …);



AXI-stream - VVC based TB (2)



UVVM (from github)

```
uvvm_util (library)
 log, check_value, await_value, etc…
```

```
bitvis_vip_clock_generator (library)
 clock_generator_vvc     (VVC)
 start_clock, ...        (procedures / methods)
 clock_generator_vvct    (global signal)
```

```
bitvis_vip_axistream (library)
 axistream_vvc                (VVC)
 axistream_transmit, ...      (procedures / methods)
 axistream_vvct               (global signal)
```

Generic to select Master or Slave

- Full UVVM (all functionality)
- Dedicated library per VVC
  - For simpler reuse
- All VIP-related functionality in dedicated VIP directories
- Script to compile all UVVM
  - Compile all, but
    Just include what you need

EmLogic

Resulting transcript +Debug

Note the changing scope

```
axistream_transmit(AXISTREAM_VVCT,0, v_data_array, msg);
```

```
ID_UVVM_SEND_CMD        50.0 ns  TB seq.(uvvm)
       ->axistream_transmit(AXISTREAM_VVC,0, 512 bytes): 'TX 512B'   [6]
```

```
ID_PACKET_DATA        24202.0 ns  AXISTREAM_VVC,0
       axistream_transmit(512B)=> Tx x"ED", byte# 493. 'TX 512B'   [6]
```

```
ID_PACKET_COMPLETE    24346.0 ns  AXISTREAM_VVC,0
       axistream_transmit(512B)=> Tx DONE. 'TX 512B'   [6]
```

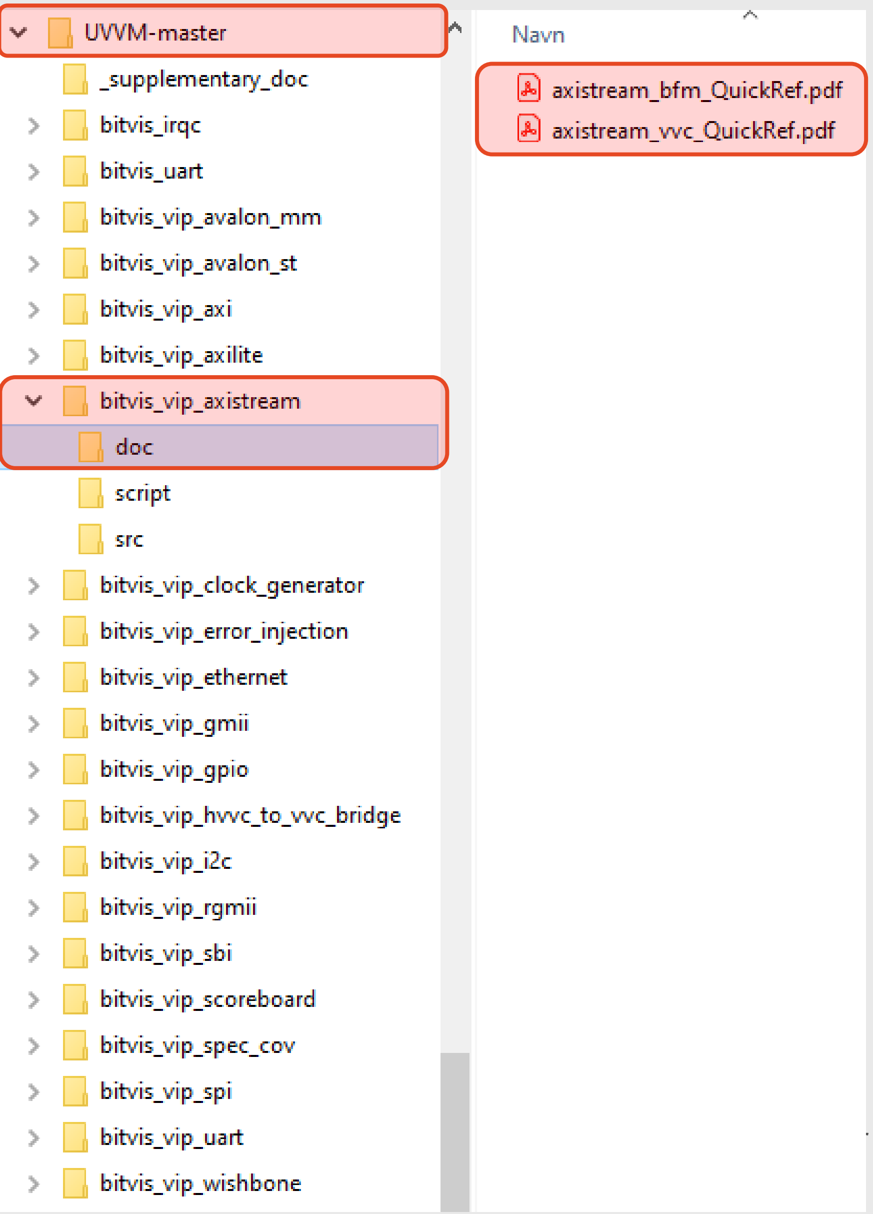
```
axistream_expect(AXISTREAM_VVCT,1, v_exp_array, "Expecting **** ");
```

```
ID_UVVM_SEND_CMD       50.0 ns   TB seq.(uvvm)
       ->axistream_expect_bytes(AXISTREAM_VVC,1, 512b): 'Expecting 512b'   [7]
```

```
-  Plus similar additional verbosity as for Transmit
-  Plus for both: Debug-messages when command reaches Interpreter and Executor
```

EmLogic

Documentation VVC



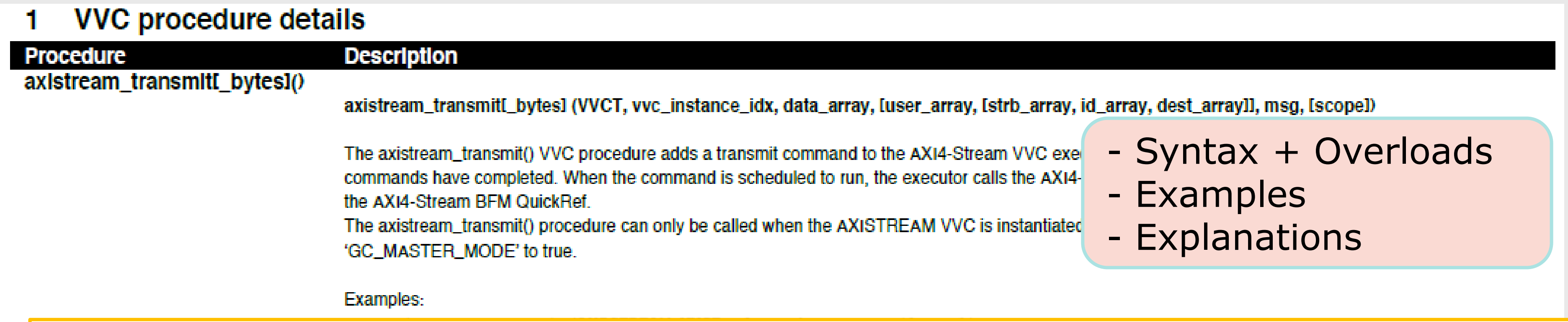
- UVVM-master
  - _supplementary_doc
  - bitvis_irqc
  - bitvis_uart
  - bitvis_vip_avalon_mm
  - bitvis_vip_avalon_st
  - bitvis_vip_axi
  - bitvis_vip_axilite
  - bitvis_vip_axistream
    - doc
    - script
    - src
  - bitvis_vip_clock_generator
  - bitvis_vip_error_injection
  - bitvis_vip_ethernet
  - bitvis_vip_gmii
  - bitvis_vip_gpio
  - bitvis_vip_hvvc_to_vvc_bridge
  - bitvis_vip_i2c
  - bitvis_vip_rgmii
  - bitvis_vip_sbi
  - bitvis_vip_scoreboard
  - bitvis_vip_spec_cov
  - bitvis_vip_spi
  - bitvis_vip_uart
  - bitvis_vip_wishbone

Navn

- axistream_bfm_QuickRef.pdf
- axistream_vvc_QuickRef.pdf

Similar docs for all BFMs, VVCs, UVVM and other VIP

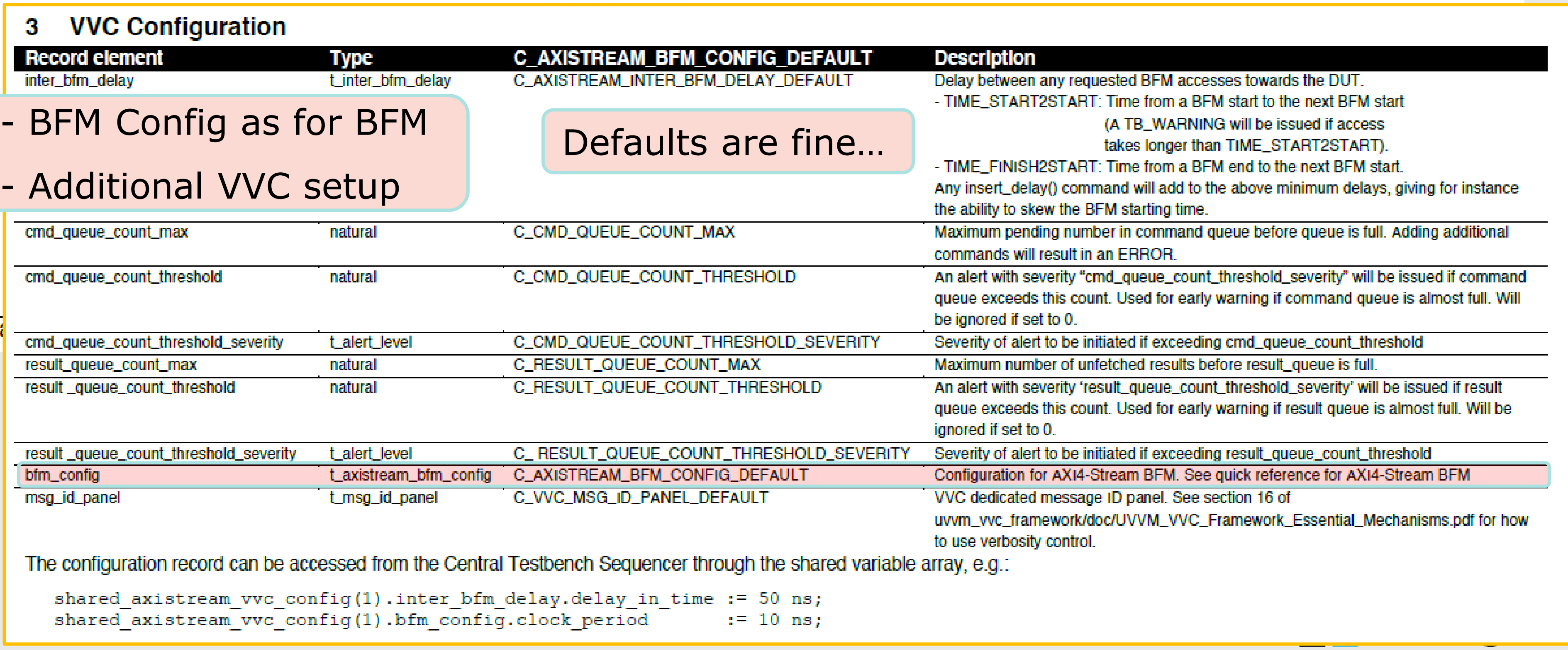
Documentation VVC



1 VVC procedure details

Procedure	Description
axistream_transmit[_bytes]()	axistream_transmit[_bytes] (VVCT, vvc_instance_idx, data_array, [user_array, [strb_array, id_array, dest_array]], msg, [scope]) The axistream_transmit() VVC procedure adds a transmit command to the AXI4-Stream VVC exe commands have completed. When the command is scheduled to run, the executor calls the AXI4- the AXI4-Stream BFM QuickRef. The axistream_transmit() procedure can only be called when the AXISTREAM VVC is instantiated 'GC_MASTER_MODE' to true. Examples:

- Syntax + Overloads
- Examples
- Explanations



3 VVC Configuration

Record element	Type	C_AXISTREAM_BFM_CONFIG_DEFAULT	Description
inter_bfm_delay	t_inter_bfm_delay	C_AXISTREAM_INTER_BFM_DELAY_DEFAULT	Delay between any requested BFM accesses towards the DUT. - TIME_START2START: Time from a BFM start to the next BFM start (A TB_WARNING will be issued if access takes longer than TIME_START2START). - TIME_FINISH2START: Time from a BFM end to the next BFM start. Any insert_delay() command will add to the above minimum delays, giving for instance the ability to skew the BFM starting time.
cmd_queue_count_max	natural	C_CMD_QUEUE_COUNT_MAX	Maximum pending number in command queue before queue is full. Adding additional commands will result in an ERROR.
cmd_queue_count_threshold	natural	C_CMD_QUEUE_COUNT_THRESHOLD	An alert with severity "cmd_queue_count_threshold_severity" will be issued if command queue exceeds this count. Used for early warning if command queue is almost full. Will be ignored if set to 0.
cmd_queue_count_threshold_severity	t_alert_level	C_CMD_QUEUE_COUNT_THRESHOLD_SEVERITY	Severity of alert to be initiated if exceeding cmd_queue_count_threshold
result_queue_count_max	natural	C_RESULT_QUEUE_COUNT_MAX	Maximum number of unfetched results before result_queue is full.
result _queue_count_threshold	natural	C_RESULT_QUEUE_COUNT_THRESHOLD	An alert with severity 'result_queue_count_threshold_severity' will be issued if result queue exceeds this count. Used for early warning if result queue is almost full. Will be ignored if set to 0.
result _queue_count_threshold_severity	t_alert_level	C_ RESULT_QUEUE_COUNT_THRESHOLD_SEVERITY	Severity of alert to be initiated if exceeding result_queue_count_threshold
bfm_config	t_axistream_bfm_config	C_AXISTREAM_BFM_CONFIG_DEFAULT	Configuration for AXI4-Stream BFM. See quick reference for AXI4-Stream BFM
msg_id_panel	t_msg_id_panel	C_VVC_MSG_ID_PANEL_DEFAULT	VVC dedicated message ID panel. See section 16 of uvvm_vvc_framework/doc/UVVM_VVC_Framework_Essential_Mechanisms.pdf for how to use verbosity control.

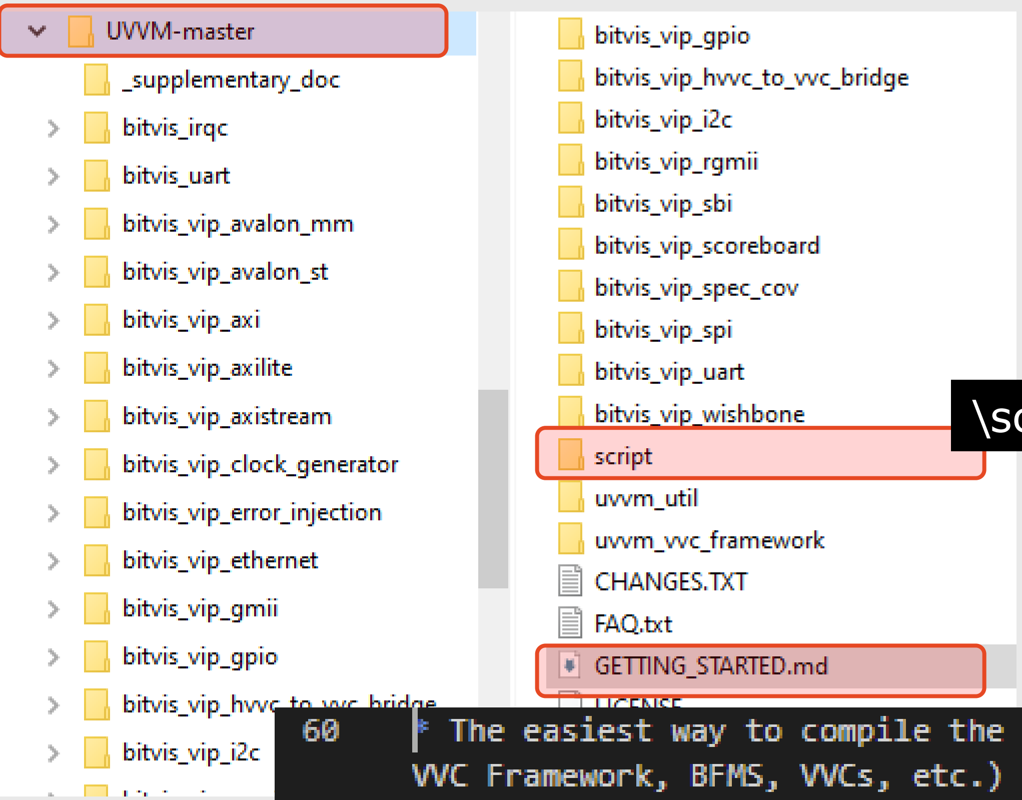
The configuration record can be accessed from the Central Testbench Sequencer through the shared variable array, e.g.:

```
shared_axistream_vvc_config(1).inter_bfm_delay.delay_in_time := 50 ns;
shared_axistream_vvc_config(1).bfm_config.clock_period       := 10 ns;
```

- BFM Config as for BFM
- Additional VVC setup

Defaults are fine…

Compiling UVVM



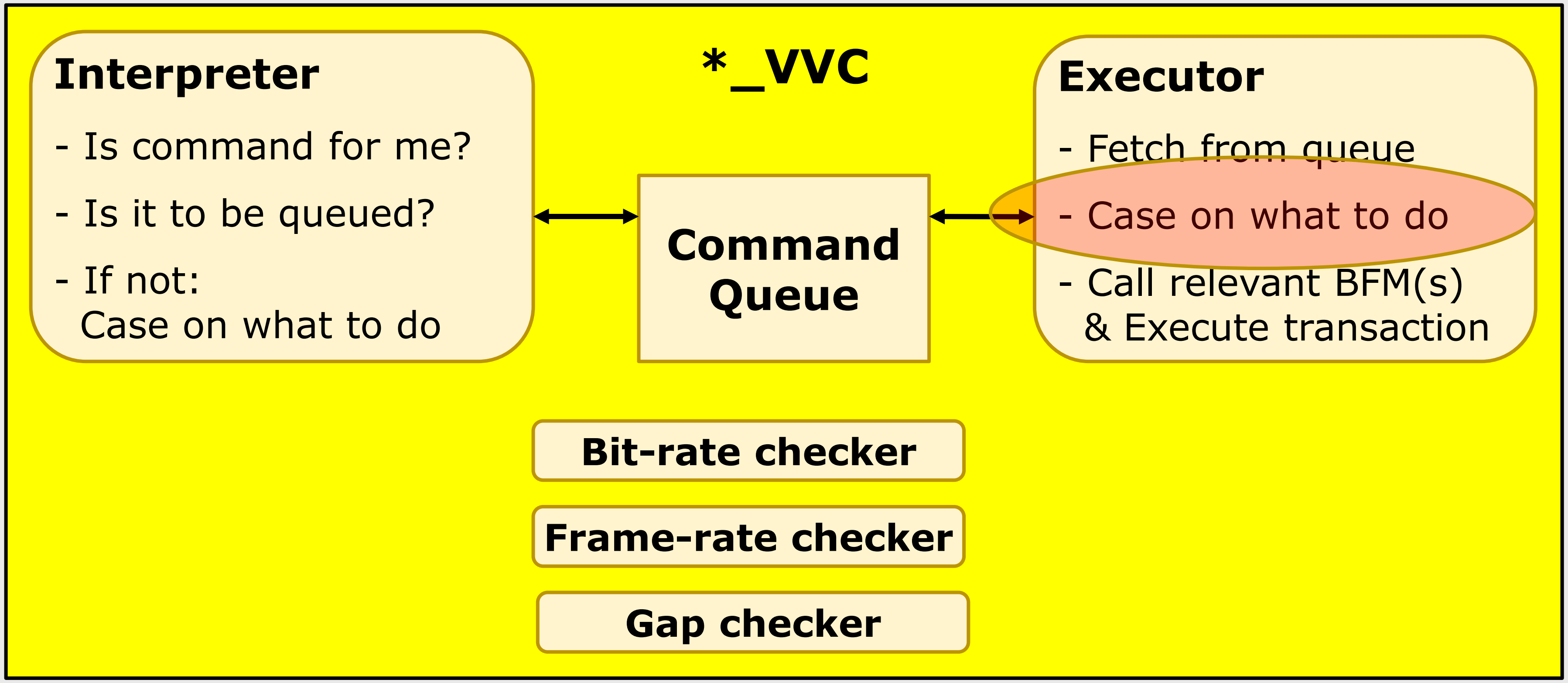
```
\script> vsim -c -do "compile_all.do"
```

```
* The easiest way to compile the complete UVVM with everything (Utility Library,
VVC Framework, BFMS, VVCs, etc.) is to go to the top-level script directory and
run 'compile_all.do' inside Modelsim/Questasim/RivieraPro/ActiveHDL.
```

EmLogic

VVC: Easy to extend

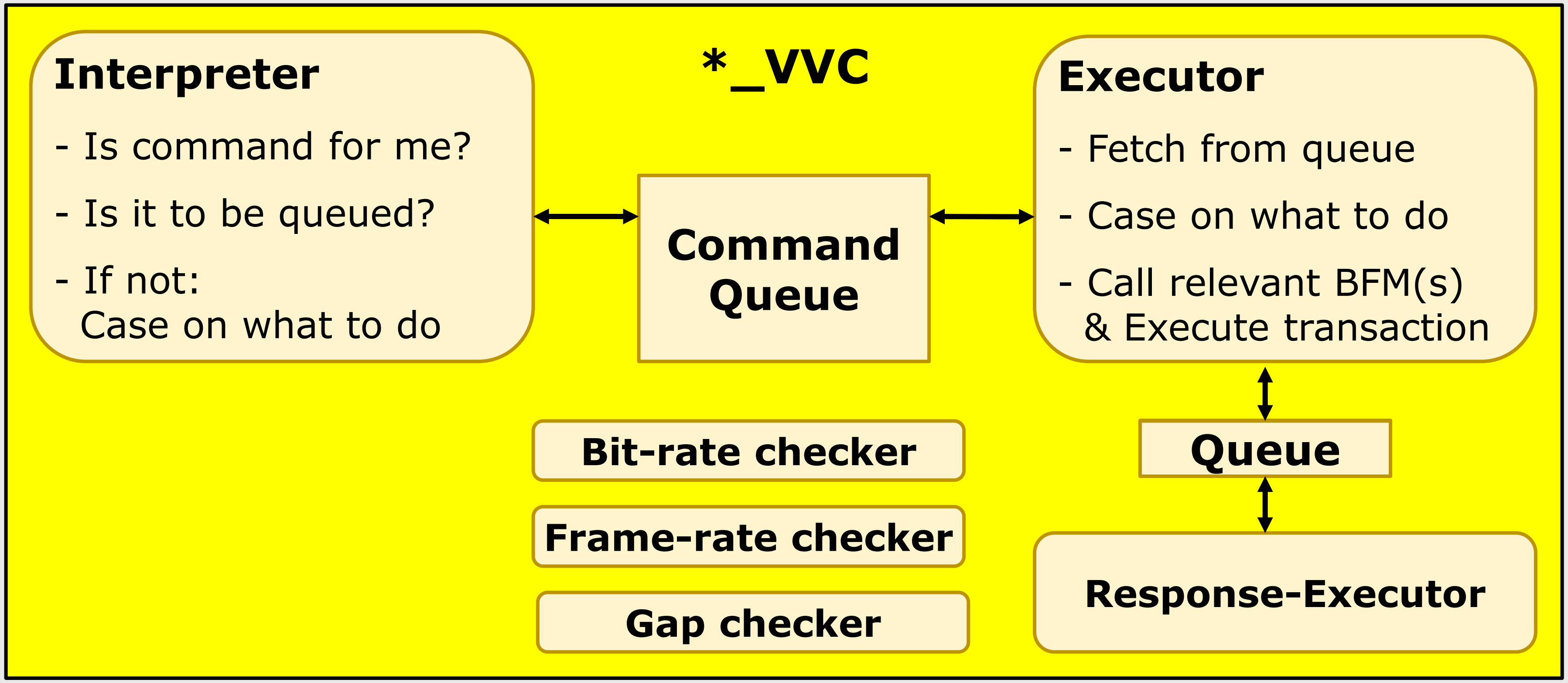
- Easy to add local sequencers
- Easy to add checkers/monitors/etc



EmLogic

VVC: Easy to extend

- **Easy to handle split transactions**
- **Easy to handle out of order execution**



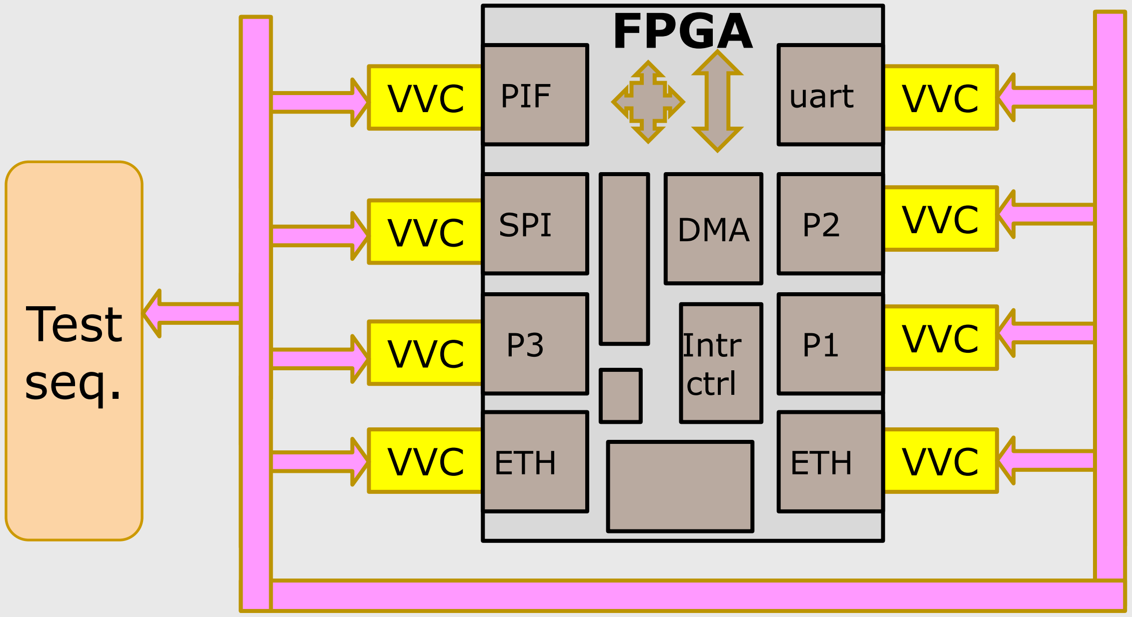
VVC Advantages

- Simultaneous activity on multiple interfaces
- Encapsulated → Reuse at all levels
- Queue → May initiate multiple high level commands
- Local Sequencers for predefined higher level commands
- **Only in UVVM VVCs:**
  - UNIQUE: Control all VVCs from a single sequencer!
  - May insert delay between commands – from sequencer
    → The only system to target cycle related corner cases
  - Simple handling of split transactions and out of order protocols
  - Common commands to control VVC behaviour
  - Simple synchronization of interface actions – from sequencer
  - May use Broadcast and Multicast

Better Overview, Maintenance, Extensibility and Reuse

Keeping the overview

- May use any number of VVCs
- May use any number of instances of each VVC type
- May control them all simultaneously – and also control command delays
- May control all from a single test sequencer (or two – or more)
- Get total overview by looking at one file of sequential commands only



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

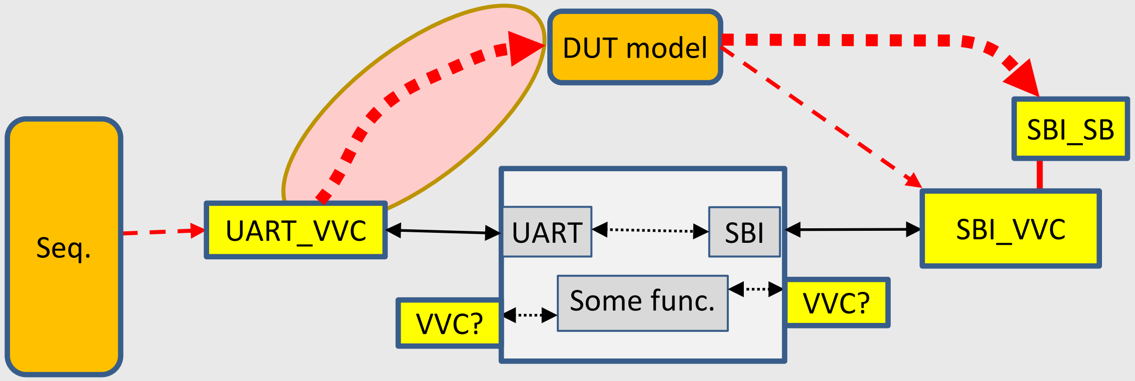
Allows:
- Simultaneous interface handling
- Synchronization of interfaces
- Skewing between interfaces
- Additional protocol checkers
- Local sequencers
- Activity detection
- Simple reuse between projects

Added 2019-20 – in cooperation with ESA

- ESA Extensions in ESA-UVVM-1
  - **Scoreboards**
  - Monitors
  - Controlling randomisation and functional coverage
  - Error injection (Brute force and Protocol aware)
  - Local sequencers
  - Controlling property checkers
  - **Watchdog** (Simple and Activity based)
  - **Transaction info**
  - Hierarchical VVCs - And Scoreboards for these
  - **Specification Coverage** (Requirement/test coverage)

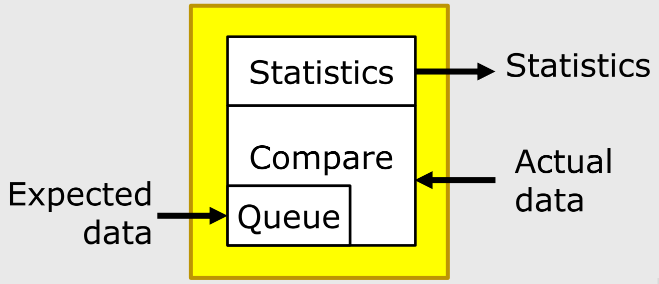
ESA is helping VHDL designers speed up FPGA and ASIC development and improve their product quality!

Transaction info transfer



Transaction info   Inside VVC   All UVVM VVCs

Generic Scoreboard



Quick Reference is provided

generic data type

- logging/reporting
- flushing queue
- clearing statistics

Configuration record:

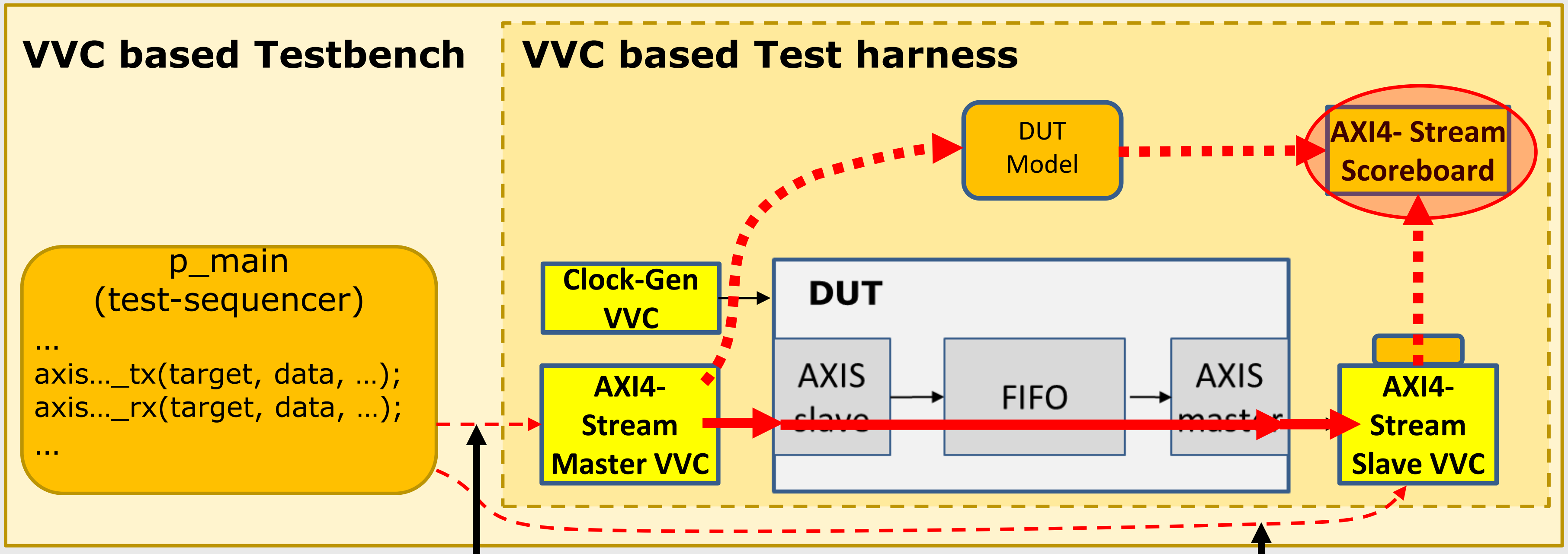
- allow_lossy
- allow_out_of_order
- mismatch_alert_level
- etc...

Counting:

- entered
- pending
- matched
- mismatched
- dropped
- deleted
- initial garbage

- insert, delete, fetch
- ignore_initial_mismatch
- indexed on either entry or position
- optional source element (in addition to expected + actual)

Advanced scoreboard-based TB



```
axistream_transmit(AXISTREAM_VVCT,0, v_data_array, msg);
```

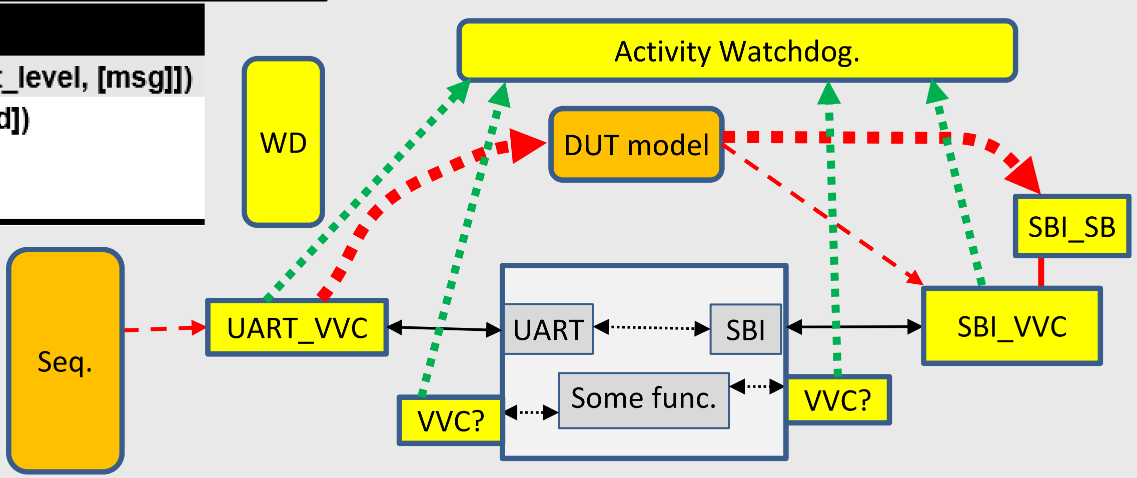
```
axistream_receive(AXISTREAM_VVCT,1, v_data_array, "Checking via SB");
```

Watchdogs

Simple WD Inside Util

Watchdog

watchdog_timer(watchdog_ctrl, timeout, [alert_level, [msg]])
extend_watchdog(watchdog_ctrl, [time_extend])
reinitialize_watchdog(watchdog_ctrl, timeout)
terminate_watchdog(watchdog_ctrl)



`Apply both concurrently`

Activity WD VVCs and UVVM

```
activity_watchdog(timeout, num_exp_vvc);
```

Specification Coverage (1)

- Assure that all requirements 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 ***

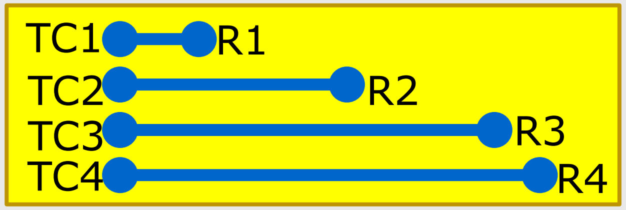
  2. Report coverage from test sequencer (or other TB parts)
  3. Generate summary report

- Solutions exist to report that a testcase finished successfully
  - BUT - reporting that a testcase has finished is not sufficient

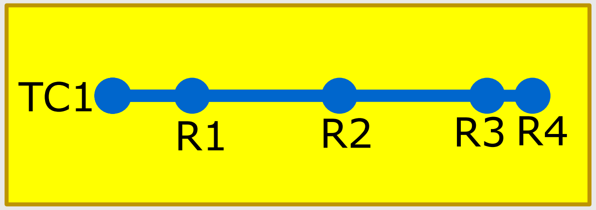
Specification Coverage (2)

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 ***

- What if multiple requirements are covered by the same testcase?
  - 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
  - Coverage per requirement
  - Test cases covering each requirement
  - Requirements covered by each Test case
- Accumulated over multiple Test cases

The new stuff – October 2021

- Enhanced Randomisation
  - Advanced randomisation in a simple way

- Optimised Randomisation
  - Randomisation without replacement
  - Weighted according to target distribution AND previous events

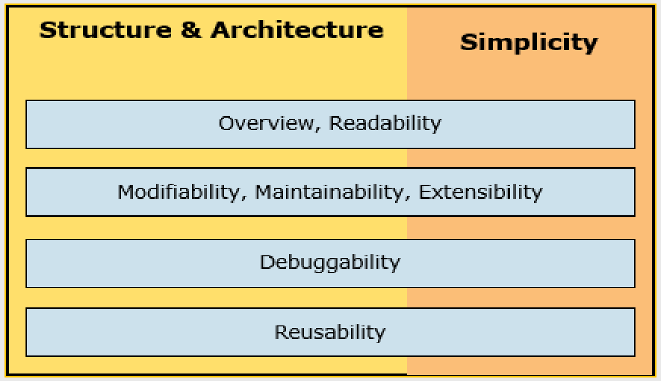
  → the lowest number of randomisations for a given target

- Functional Coverage
  - Based on functional coverage in SV
    - But in VHDL, and without all the complexity of SV and UVM
  - Fully integrated with UVVM, but may be used stand-alone

UVVM Enhanced Randomisation

- Well integrated with UVVM
  - Alert handling and logging in particular
- Strong focus on Overview & Readability
  - Adding keywords to ease understanding
- Easy to Maintain and Extend

Quality & Efficiency enablers



Typing code consumes is an insignificant part of the development time.

Reading and understanding code is repeated over and over again, and is thus a significant part of the development time

```
addr <= my_addr.rand(0, 18, ADD,(30,31), EXCL,(7));
```

➔ **Investing in better code yields a huge return on investment**

Single Method approach

- **"Standard" approach: Randomisation in one single command**
  - Simple randomisation is always easy to understand

```
addr <= my_addr.rand(0, 18);
```

  - More complex randomisation is normally more difficult to understand
    BUT – there are ways to significantly improve this

```
addr <= my_addr.rand(0, 18, EXCL,(7));
```

```
addr <= my_addr.rand(0, 18, ADD,(30,31));
```

```
addr <= my_addr.rand(0, 18, ADD,(30,31), EXCL,(7));
```

  - Similar readability focus for weighting

```
addr <= my_addr.rand_val_weight((0,2),(1,3),(2,5));
```

```
addr <= my_addr.rand_range_weight((0,18,4),(19,31,1));
```

EmLogic

Multi-method approach (1)

- Extends the functionality of the single method approach
  - Single method approach:

```
addr_1 <= my_addr.rand(0, 18, ADD,(30,31), EXCL,(7));
addr_2 <= my_addr.rand(0, 18, ADD,(30,31), EXCL,(7));
```

  - Multi-method - equivalent

```
my_addr.add_range(0, 18);
my_addr.add_val((30,31));
my_addr.excl_val((7));
addr_1 <= my_addr.randm(VOID);
addr_2 <= my_addr.randm(VOID);
```

Note: rand**m**()
(For clarity
and to avoid any ambiguity)

  - Allows adding more ranges, sets or exclusions

```
my_addr.add_range(48,63);
my_addr.add_range(80,127);
```

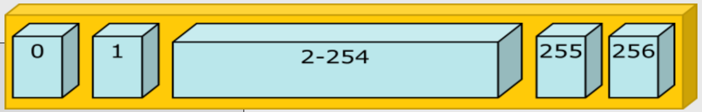
  - Allows simple inclusion of future extensions

Functional Coverage – Typical Sequence

- Define a variable of type t_coverpoint

```
variable cp_payload_size : t_coverpoint;
```

- Add the bins



```
cp_payload_size.add_bins(bin(0));
cp_payload_size.add_bins(bin(1));
cp_payload_size.add_bins(bin_range(2,254,1));
cp_payload_size.add_bins(bin(255,256,2));
```

- Tick off bins as their corresponding payload size is used

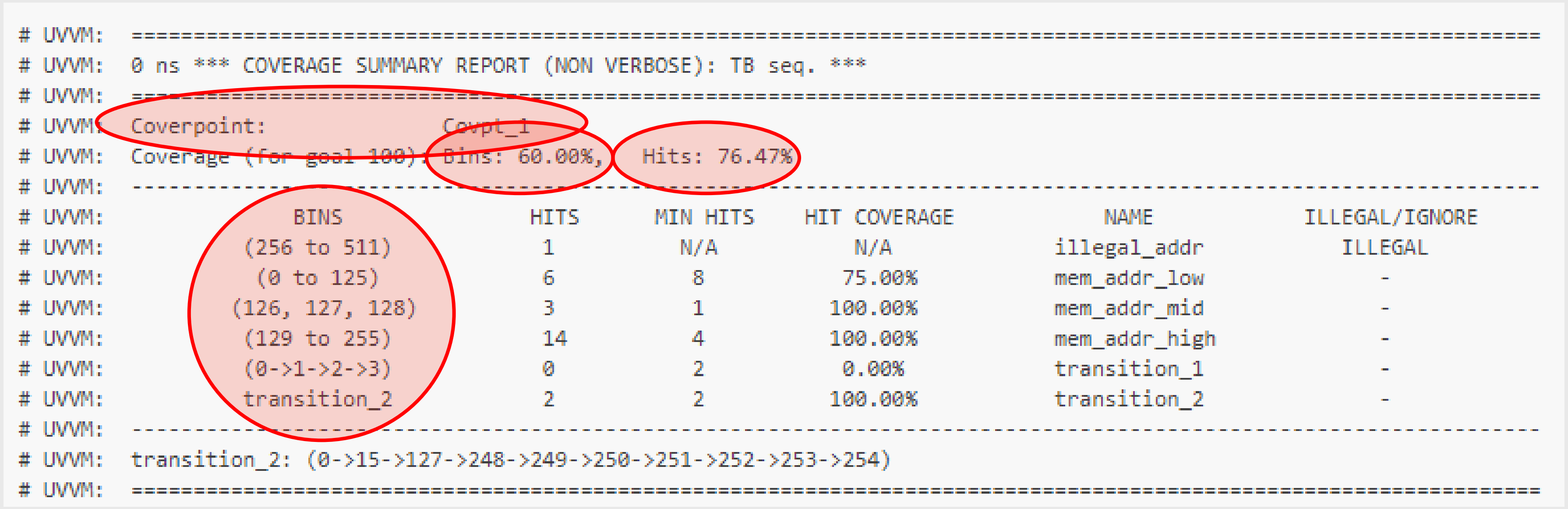
```
cp_payload_size.sample_coverage(payload_size);
```

- Continue sending packets until coverage target is reached

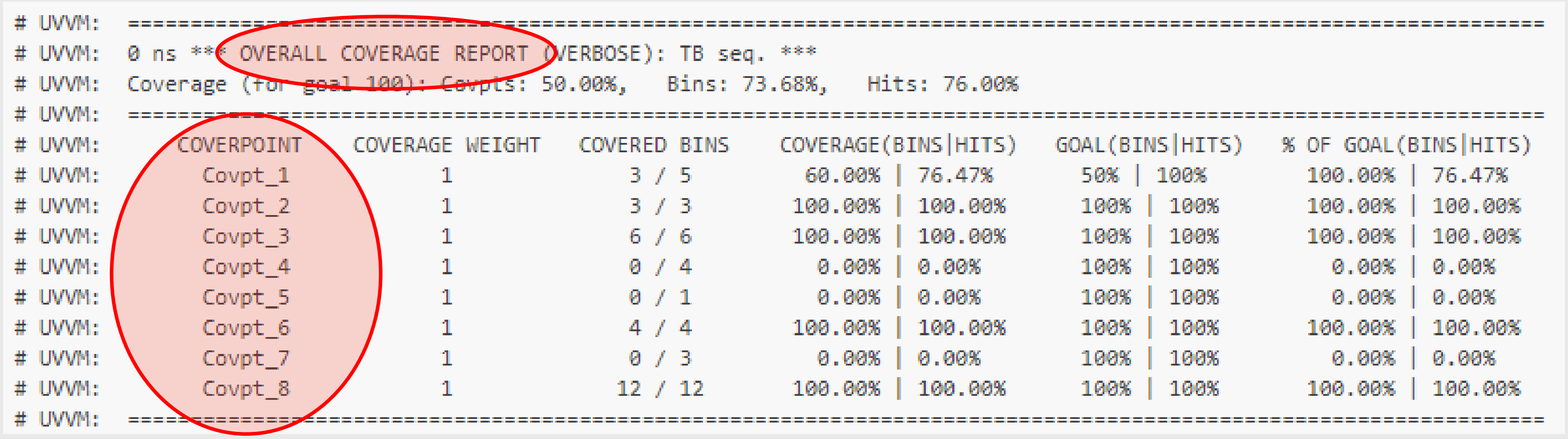
```
while not cp_payload_size.coverage_completed(VOID);
```

UVVM also has transition coverage

Some reports – out of many



```
# UVVM:  ==================================================================================================
# UVVM:  0 ns *** COVERAGE SUMMARY REPORT (NON VERBOSE): TB seq. ***
# UVVM:  ==================================================================================================
# UVVM:  Coverpoint:              Covpt_1
# UVVM:  Coverage (for goal 100): Bins: 60.00%,    Hits: 76.47%
# UVVM:  --------------------------------------------------------------------------------------------------
# UVVM:             BINS                HITS     MIN HITS    HIT COVERAGE          NAME           ILLEGAL/IGNORE
# UVVM:          (256 to 511)            1          N/A          N/A          illegal_addr          ILLEGAL
# UVVM:           (0 to 125)             6          8           75.00%        mem_addr_low             -
# UVVM:         (126, 127, 128)          3          1          100.00%        mem_addr_mid             -
# UVVM:          (129 to 255)           14          4          100.00%        mem_addr_high            -
# UVVM:          (0->1->2->3)            0          2            0.00%        transition_1             -
# UVVM:          transition_2            2          2          100.00%        transition_2             -
# UVVM:  --------------------------------------------------------------------------------------------------
# UVVM:  transition_2: (0->15->127->248->249->250->251->252->253->254)
# UVVM:  ==================================================================================================
```



```
# UVVM:  ==================================================================================================
# UVVM:  0 ns *** OVERALL COVERAGE REPORT (VERBOSE): TB seq. ***
# UVVM:  Coverage (for goal 100): Covpts: 50.00%,   Bins: 73.68%,   Hits: 76.00%
# UVVM:  ==================================================================================================
# UVVM:     COVERPOINT     COVERAGE WEIGHT    COVERED BINS    COVERAGE(BINS|HITS)    GOAL(BINS|HITS)    % OF GOAL(BINS|HITS)
# UVVM:      Covpt_1              1               3 / 5         60.00% | 76.47%       50% | 100%        100.00% | 76.47%
# UVVM:      Covpt_2              1               3 / 3        100.00% | 100.00%     100% | 100%        100.00% | 100.00%
# UVVM:      Covpt_3              1               6 / 6        100.00% | 100.00%     100% | 100%        100.00% | 100.00%
# UVVM:      Covpt_4              1               0 / 4          0.00% | 0.00%       100% | 100%          0.00% | 0.00%
# UVVM:      Covpt_5              1               0 / 1          0.00% | 0.00%       100% | 100%          0.00% | 0.00%
# UVVM:      Covpt_6              1               4 / 4        100.00% | 100.00%     100% | 100%        100.00% | 100.00%
# UVVM:      Covpt_7              1               0 / 3          0.00% | 0.00%       100% | 100%          0.00% | 0.00%
# UVVM:      Covpt_8              1              12 / 12       100.00% | 100.00%     100% | 100%        100.00% | 100.00%
# UVVM:  ==================================================================================================
```

EmLogic

Pick and choose – No lock

- Pick **any** Utility Library functionality: (from these plus more)

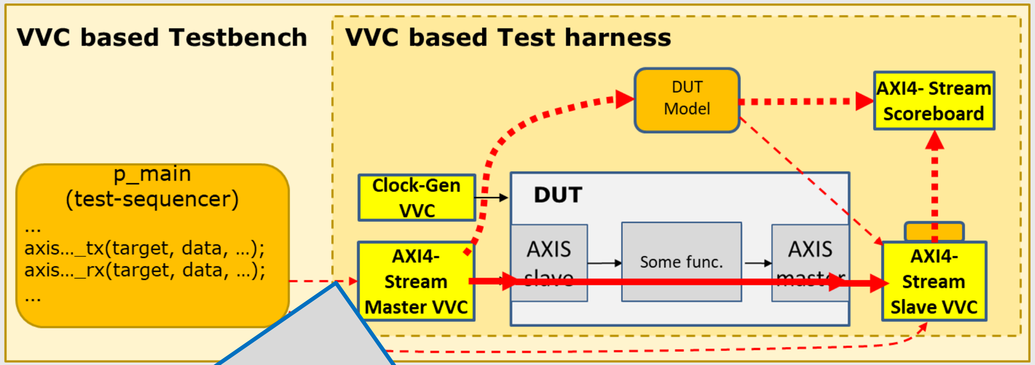
`log()` `alert()` `error()` `manual_check()`
`check_value()` `check_stable()` `await_stable()`
`await_change()` `await_value()` `check_value_in_range()`
`random()` `randomize()` `report_***()` `enable_log_msg()`
`justify()` `fill_string()` `to_upper()` `replace()`
`clock_generator()` `await_unblock_flag()` `await_barrier()`

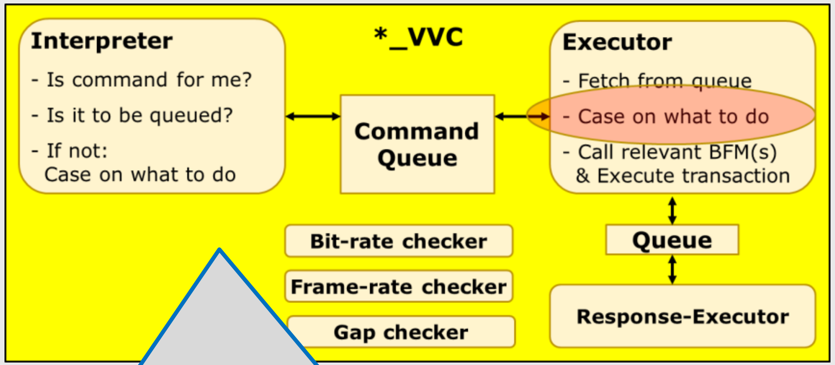
- Pick any BFM - or any VVC – or any combination

`AXI4-lite` `GPIO` `SBI` `SPI` `UART` `I2C`
`AXI` `AVALON MM` `AXI4-stream` `Avalon-stream`
`CLOCK_GENERATOR` `GMII` `RGMII` `Ethernet`

- Pick any FIFO, Queue, Scoreboard
- Pick any Advanced Randomisation and/or Functional Coverage
- Pick Specification coverage / Requirements tracking

Standardized? – In what way?





- Standard Interface
- Standard Protocol
- Standard common commands
- Standard Status interface
- Standard Config interface
- Standard handling of multiple VVCs
- Standard VVC synchronization
- Standard multicast/broadcast

- Standard VVC internal architecture
- Standard VVC control of checkers
- Standard queuing system
- Standard handling of multi-threaded interfaces
- Standard debug support

Simplification

VVCs from different users will work together

Users know how VVCs behave and how any test harness will work

UVVM vs UVM

- UVVM: VHDL (2008) vs UVM: SystemVerilog
- UVVM: Component oriented vs UVM: Object oriented
- Block diagrams are similar, but different naming and structure

- UVM is far more comprehensive and complex than UVVM
  - But UVVM is sufficient for almost all testbenches
- UVVM user threshold is a fraction of the UVM threshold – for VHDL users
  - UVVM is just a step-by-step evolution on VHDL
- UVVM allows a gentle introduction to modern verification
  - **May** be used as a first step to UVM – for those who evaluate that
  - Is however sufficient in itself for almost all FPGA designs
- UVVM can run on any VHDL 2008 compatible simulator

Courses

- FPGA (and ASIC) Verification:
  'Advanced VHDL Verification – Made simple'
- FPGA (and ASIC) Design:
  'Accellerating FPGA and Digital ASIC Design'

Design

- Design Architecture & Structure
- Clock Domain Crossing
- Coding and General Digital Design
- Reuse and Design for Reuse
- Timing Closure
- Quality Assurance - at the right level
- Faster and safer design

Verification

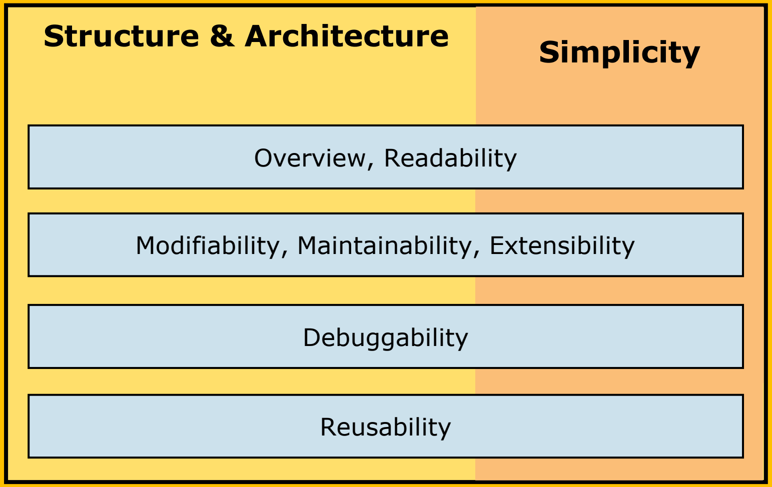
- Verification Architecture & Structure
- Self checking testbenches
- BFMs – How to use and make
- Checking values, time aspects, etc
- Verification components
- Advanced Verif: Scoreboard, Models, etc
- State-of-the-art verification methodology

Next courses in Germany October and November.
More courses on demand: On-site, Online, Public, or Hybrid

https://emlogic.no/courses/

UVVM in a nutshell

- Huge improvement potential for more structured FPGA verification



Structure & Architecture
Simplicity
Overview, Readability
Modifiability, Maintainability, Extensibility
Debuggability
Reusability

UVVM (incl. all) is Open Source

Game changer for efficiency & quality

UVVM has the largest collection of interface models (as BFMs and VVCs)

UVVM may save 200-2000 hours on a medium complex project

And at the same time improve TTM, MTBF & LCC

Usage is exploding

- World-wide number 1 for VHDL
- Fastest growing – of all