




# EmLogic

## **MODERN VHDL TESTBENCHES**

AN AXI-STREAM EXAMPLE,  
FIRST dead simple, - THEN advanced  
- Both as simple as possible

***FPGA Conference Europe, Live Online, 6 July 2021***

- Independent Design Centre for Embedded Systems and FPGA
- Established 1<sup>st</sup> of January 2021. **Extreme ramp up**
  - January: 1 person
  - August : → 16 designers (SW:6, HW:1, FPGA:9) - **And still growing fast...**
- Continues the legacy from  **bitvis**
  - All 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)

# What is UVVM?

UVVM = Universal VHDL Verification Methodology

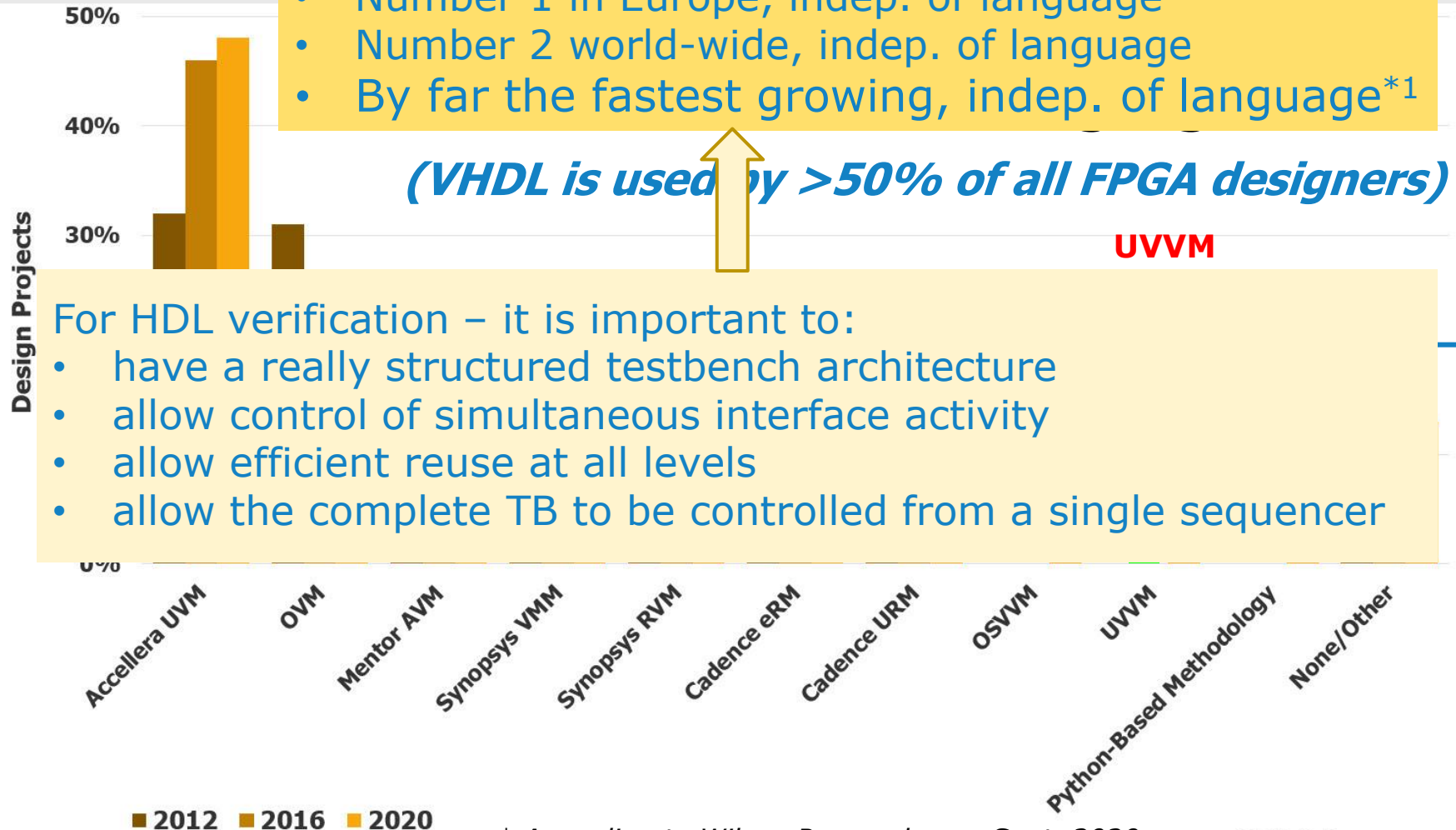
- Open Source Verification Library & Methodology
- Very structured infrastructure and architecture
- Significantly improves Verification Efficiency
- Assures a far better Design Quality
- Unique Reuse friendliness
- Recommended by Doulos for Testbench architecture
- Supported by more and more EDA vendors
- ESA projects to extend the functionality
- Extremely fast adoption by the world-wide VHDL community



# UVVM – World-wide #1

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

*(VHDL is used by >50% of all FPGA designers)*



For HDL verification – it is important to:

- have a really structured testbench architecture
- allow control of simultaneous interface activity
- allow efficient reuse at all levels
- allow the complete TB to be controlled from a single sequencer

\* According to Wilson Research, per Sept. 2020

\*\* Multiple answers possible

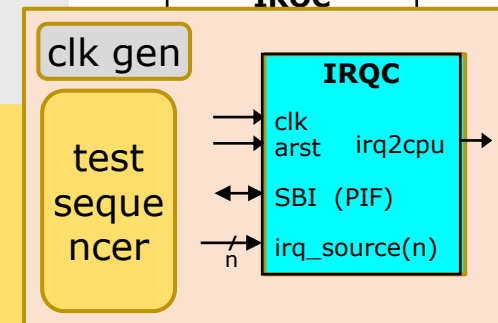
# Typical simple verif. scenario

## - a low complexity interrupt controller

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

```
log(ID_LOG_HDR, "Started simulation of IRQC_TB");  
...  
check_value(irq2cpu, '0', "irq2cpu default inactive");  
...  
check_stable(irq2cpu, now - v_reset_time);  
...  
gen_pulse(irqc_source(2), '1', clk_period, "Set source 2 for clock period");  
gen_pulse(irqc_source(3), '1', clk, 1, "Set source 3 for 1 period");  
...  
await_value(irq2cpu, '1', 0 ns, 2* C_CLK_PERIOD,  
            "Interrupt expected immediately");  
...  
sbi_write(C_ADDR_ITR, x"AA", "ITR : Set interrupts");  
sbi_check(C_ADDR_IRR, x"AA", "IRR");  
sbi_write(C_ADDR_ITR, x"55", "ITR : Set more interrupts");  
sbi_check(C_ADDR_IRR, x"FF", "IRR");  
...  
report_alert_counters(FINAL);
```

### Testbench



### All procedures with:

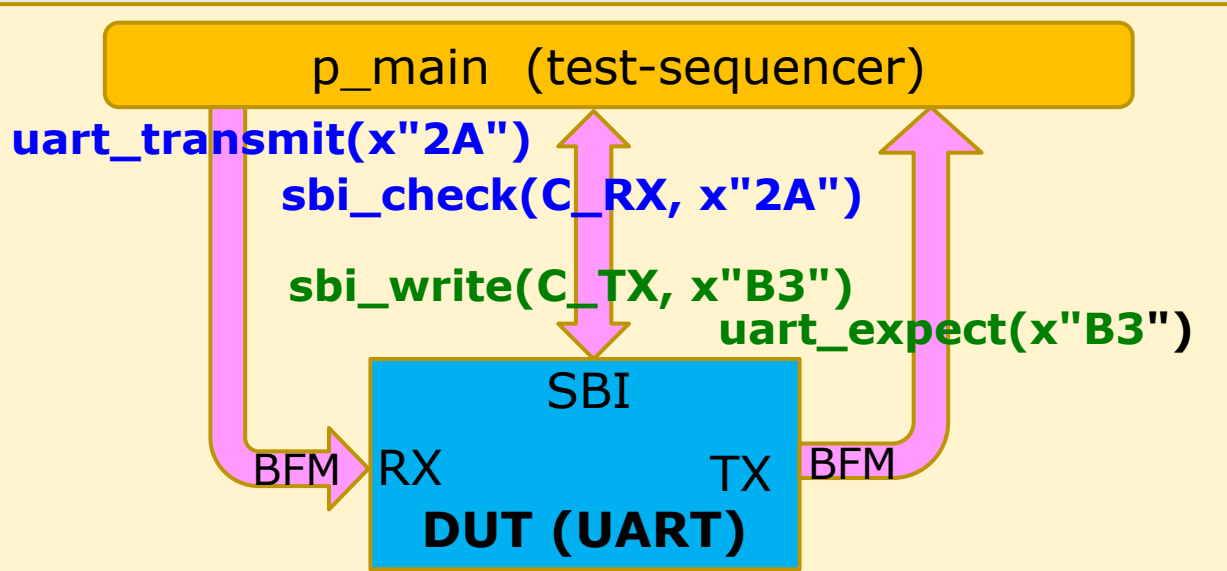
- Positive acknowledge  
If wanted
- Alert message  
and mismatch report
- Alert count and ctrl

# More in UVVM Utility Library

- `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...

Name	Parameters and examples	Description
[v_bool :=] check_value()	<pre> val(bool), [exp(bool)], alert_level, msg, [scope, [msg_id, [msg_id_panel]]] val(slv), exp(slv), [match_strictness], alert_level, msg, [scope, [msg_id, [msg_id_panel]]] val(slv), exp(slv), [match_strictness], alert_level, msg, [scope, [radix, [format, [msg_id, [msg_id_panel]]]]] val(u), exp(u), alert_level, msg, [scope, [radix, [format, [msg_id, [msg_id_panel]]]]] val(s), exp(s), alert_level, msg, [scope, [radix, [format, [msg_id, [msg_id_panel]]]]] val(int), exp(int), alert_level, msg, [scope, [msg_id, [msg_id_panel]]] val(real), exp(real), alert_level, msg, [scope, [msg_id, [msg_id_panel]]] val(time), exp(time), alert_level, msg, [scope, [msg_id, [msg_id_panel]]]  <b>Examples</b> check_value(v_int_a, 42, WARNING, "Checking the integer"); v_check := check_value(v_slv5_a, "11100", MATCH_EXACT, ERROR, "Checking the SLV", "My Scope",                       HEX, AS_IS, ID_SEQUENCER, shared_msg_id_panel); </pre>	<p>Checks if <i>val</i> equals <i>exp</i>, and if the values do not match.</p> <p>The result of the check is returned as a function.</p> <p>If <i>val</i> is of type <i>slv</i>, unsigned arguments:</p> <ul style="list-style-type: none"> <li>- <i>match_strictness</i>: Specifies the match strictness, e.g. MATCH_EXACT.</li> <li>- <i>radix</i>: for the vector representation, e.g. HEX, BIN, IF_INVALID.</li> <li>- <i>format</i>: for the vector representation, e.g. U, X, Z or W, - in which the vector contains any.</li> <li>- <i>format</i> may be AS_IS or formatted in the log.</li> </ul>
[tb_]error(msg, [scope])  [tb_]failure(msg, [scope])		Randomize(seed1, seed2)  <b>Signal generators</b>

# Data communication



May use Utility Library  
and provided BFM

```
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:=====
```

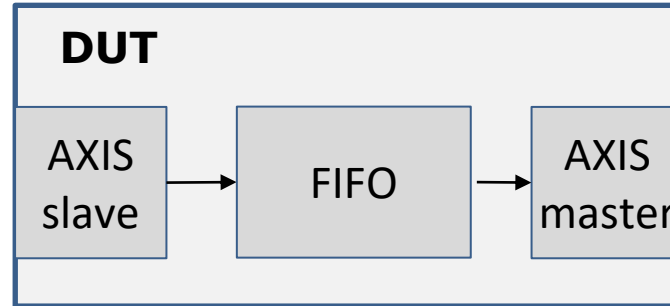


# UVVM Download

- UVVM – The full UVVM  
<https://github.com/UVVM/UVVM>
  - Contains the full UVVM with everything you need
    - ◆ Utility Library, all BFM, VVC framework, all VVCs, additional general VIP
  - All VVCs/VIP are located in dedicated libraries and directories
  - Dedicated scripts to compile all or parts
- UVVM-Light – A subset of the full UVVM - without VVCs  
[https://github.com/UVVM/UVVM\\_Light](https://github.com/UVVM/UVVM_Light)
  - Contains everything you need if you do not want VVCs or Advanced VIP
    - ◆ Utility Library, all BFM
  - Utility library and all BFM located in one single library and directory
  - Dedicated script to compile all
  - Was provided on request from novice designers who
    - did not properly understand how to handle multiple libraries,
    - wanted fewer files and a smaller footprint
- May be Cloned directly or Downloaded as a ZIP-file – Directly from Github

# AXI Stream (AXIS) DUT ++

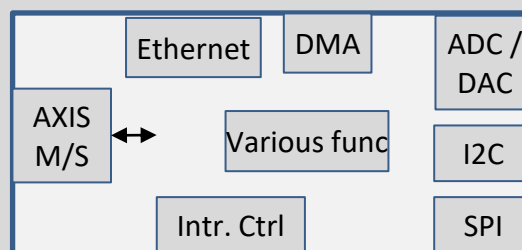
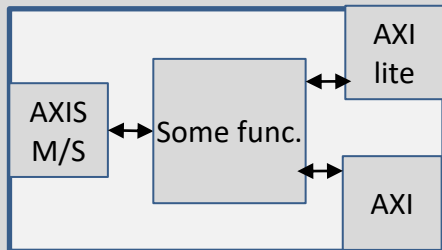
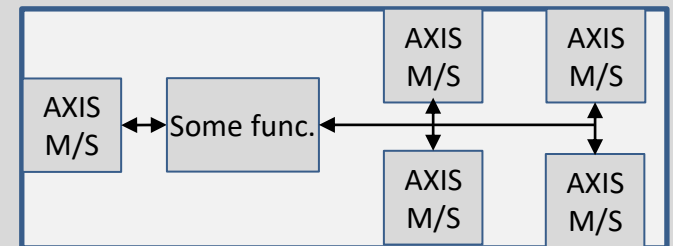
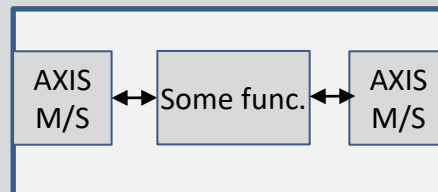
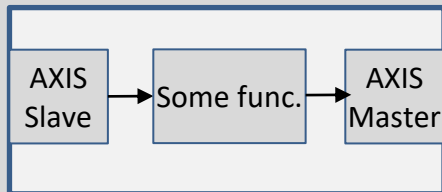
Example DUT:



Starting point:

- tdata 8-bit
- tvalid
- tready

Other DUT scenarios are handled much the same way:



etc....

# AXI-stream - BFM based TB

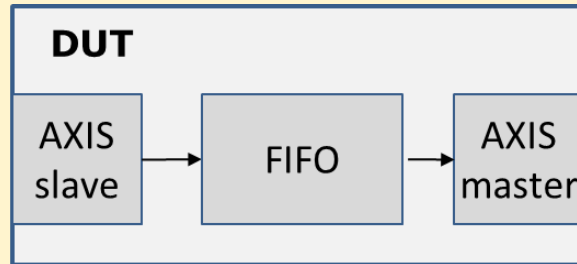
## - as simple as possible

clock\_generator

p\_main  
(test-sequencer)

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

**BFM based Testbench**



**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...
```

- No test harness (for simplicity)
- Sequencer has direct access to DUT signals
  - Thus BFM from p\_main can also see the DUT signals
- Simplified UVVM
  - For simple usage
- Subset of UVVM  
No VVCs or VCC support
- All BFM in the same directory and library

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

# Required code for AXI-Stream BFM (Using UVVM\_Light)

- Need to include library and packages in TB code:
- Define your DUT-dedicated AXI-stream record:  
(A must for pre-defined records)
- Define your signals:  
(Mandatory in any case)
- Set tkeep = '1' to the slave BFM  
(Must indicate some way...)
- **You are now ready to write any sequence of transmit, receive or expect:**

```
library uvvm_util;  
context uvvm_util.uvvm_util_context;  
use uvvm_util.axistream_bfm_pkg.all;
```

```
subtype t_axis is t_axistream_if(  
    tdata(7 downto 0), tkeep(0 downto 0),  
    tuser(0 downto 0), tstrb(0 downto 0),  
    tid(0 downto 0),    tdest( 0 downto 0));
```

```
signal m_axis  : t_axis;  
signal s_axis  : t_axis;
```

```
s_axis.tkeep <= "1";  
-- s_axis.tkeep <= (others => '1');
```

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

```
axistream_expect  
  (v_exp_data(32 to 63), msg, clk, s_axis);
```

- **Or local overloads** (skipping the signal parameters):

```
axis_transmit(v_byte_array, msg);
```

```
axis_expect(v_exp_data(32 to 63), msg);
```

```
axis_transmit(("D0", "D1", "D2", "D3"), msg);
```

# 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.
```

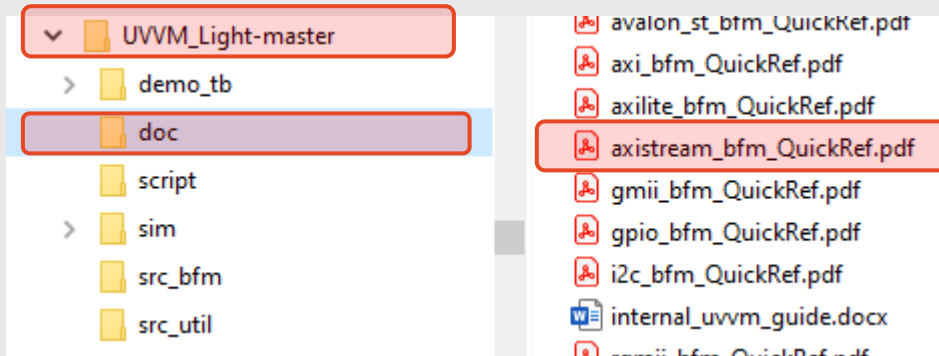
## May add more info for debugging

```
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



Similar docs for all BFMs

# Documentation BFM

## AXI4-Stream BFM – Quick Reference

- Syntax + Overloads
- Examples
- Explanations

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

**axistream\_transmit[bytes]** (data\_array, [user\_array, [strb\_array, id\_array, dest\_array]], msg, clk, axistream\_if, [scope, [msg\_id\_panel, [config]]])

Example (tdata'length = 16) : axistream\_transmit ( (x"D0", x"D1", x"D2", x"D3"), (x"00", x"0A"), "Send a 4 byte packet with tuser=A at the 2<sup>nd</sup> (last) word", clk, axistream\_if);  
Example (tdata'length = 8) : axistream\_transmit ( (x"D0", x"D1", x"D2", x"D3"), (x"00", x"00", x"00", x"0A"), "Send a 4 byte packet with tuser=A at the 4<sup>th</sup> (last) word", clk, axistream\_if);

Example: axistream\_transmit(v\_data\_array(0 to v\_numByte-1), v\_user\_array(0 to v\_numByte-1), v\_strb\_array(0 to v\_numByte-1), v\_id\_array(0 to v\_numByte-1), v\_dest\_array(0 to v\_numByte-1), msg, clk, axistream\_if, scope, msg\_id\_panel, config);  
Example: axistream\_transmit(v\_data\_array(0 to v\_numByte-1), v\_user\_array(0 to v\_numByte-1), v\_strb\_array(0 to v\_numByte-1), v\_id\_array(0 to v\_numByte-1), v\_dest\_array(0 to v\_numByte-1), msg, clk, axistream\_if, scope, msg\_id\_panel, config);  
Example: axistream\_transmit(v\_data\_array(0 to v\_numByte-1), v\_user\_array(0 to v\_numByte-1), v\_strb\_array(0 to v\_numByte-1), v\_id\_array(0 to v\_numByte-1), v\_dest\_array(0 to v\_numByte-1), msg, clk, axistream\_if, scope, msg\_id\_panel, config);

Note! Use axistream\_transmit\_bytes ( ) when using t\_byte.

### BFM Configuration record 't\_axistream\_bfm\_config'

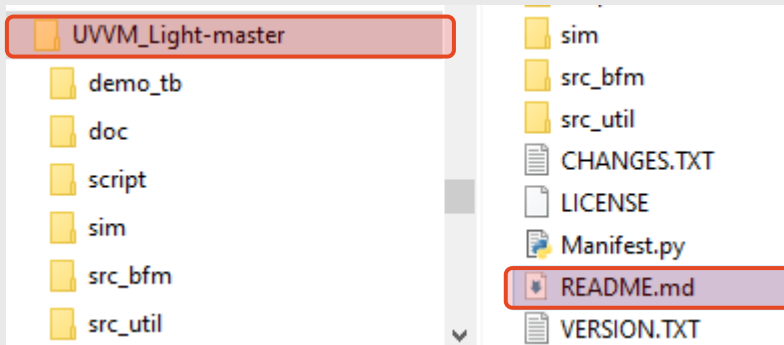
Record element	Type	C_AXISTREAM_BFM_CONFIG_DEFAULT
max_wait_cycles	integer	100
max_wait_cycles_severity	t_alert_level	ERROR
clock_period	time	-1 ns
clock_period_margin	time	0 ns
clock_margin_severity	t_alert_level	TB_ERROR
setup_time	time	-1 ns
hold_time	time	-1 ns
bfm_sync	t_bfm_sync	SYNC_ON_CLOCK_ONLY
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

Defaults are fine...

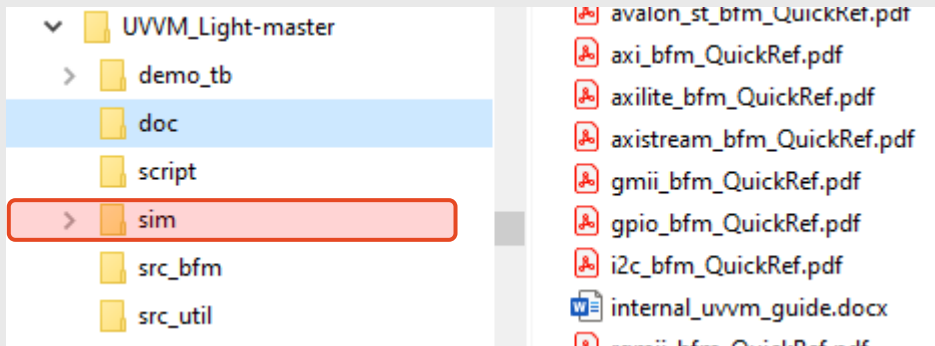
### Configuration

- Protocol Behaviour
- Compliance checking
- Simulation set-up

# Compiling UVVM Light



```
24 E.g. compiling from the /sim folder inside the UVVM Light install directory:
25  `sh
26  $ vsim -c -do "do ../script/compile.do [uvvm_light_directory] [target_directory]"
```



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



# Advanced BFM usage - in simple TB

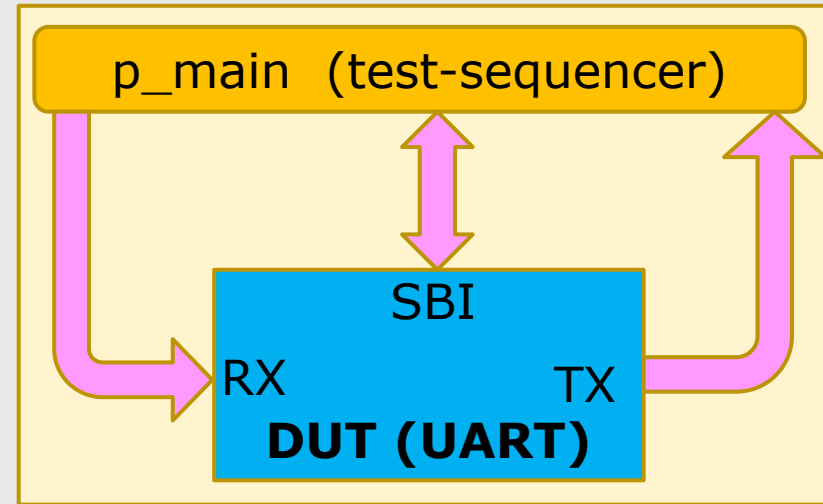
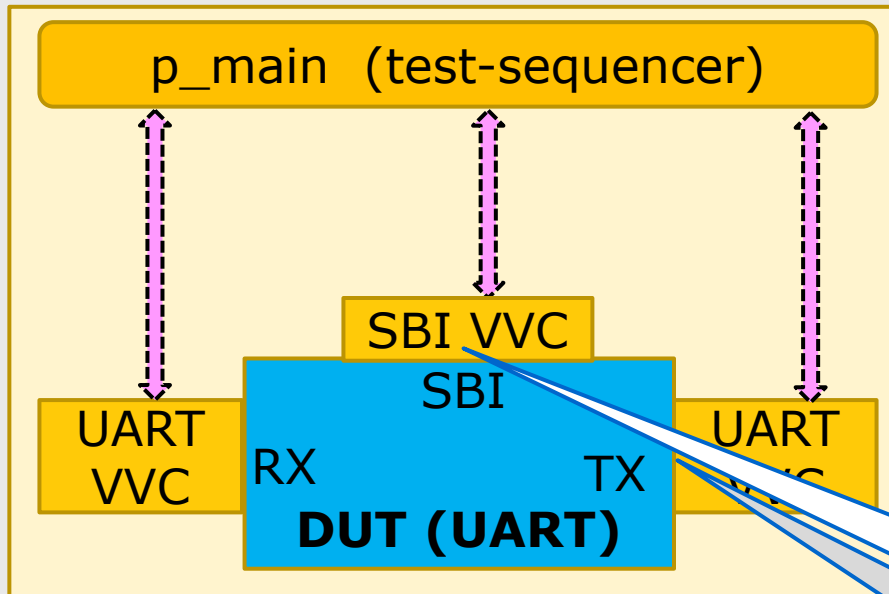
- May utilise more of the protocol:
- May define different widths
- May configure behaviour:
  - Set maximum wait cycles
  - May set to match data exact or std\_match
  - May set byte endianness (for SLV larger than data width)
  - May set to de-assert tvalid some cycles (randomly or fixed)
  - May set to de-assert tready some cycles (randomly or fixed)
  - And more...

tkeep, tuser, tlast,  
tstrb, tid, tdest

**Have enabled lots of bug detection in users' AXI stream interfaces**

valid_low_at_word_num	Word index during which the Master BFM shall deassert valid while sending a packet.
valid_low_duration	Number of clock cycles to deassert valid.
valid_low_multiple_random_prob	Similar for 'ready'
valid_low_max_random_duration	

# For more advanced DUT complexity: → Use VVCs

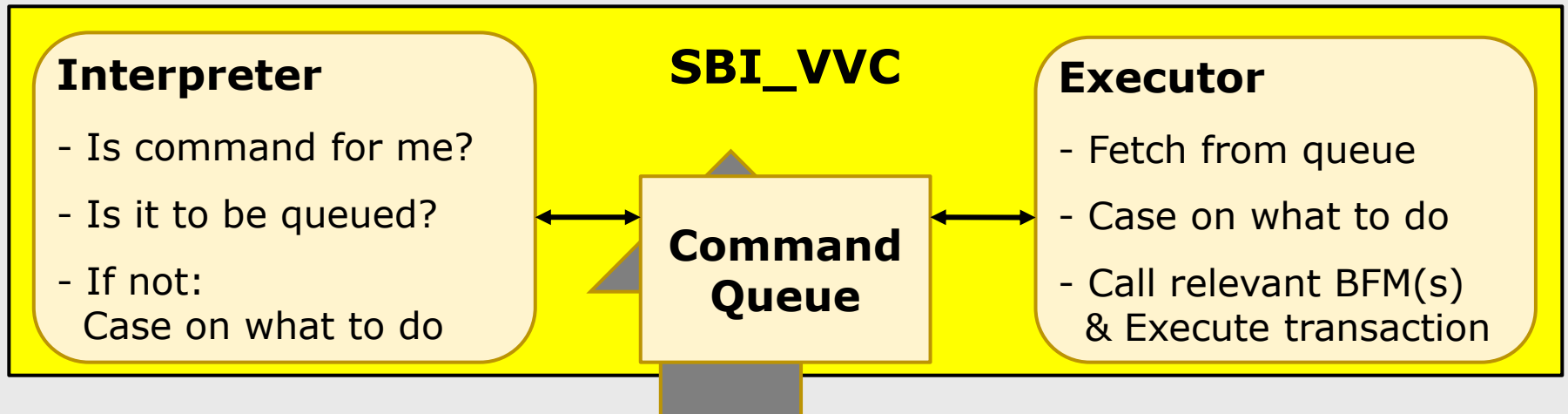


```
sbi_write(SBI_VVCT,1, C_TX, x"B3")  
uart_expect(UART_VVCT, 1, RX, x"B3")
```

```
sbi_write(C_TX, x"B3")
```

```
uart_expect(x"B3")
```

# VVC: VHDL Verification Component



## Same main architecture in every VVC

- >95% same code - apart from BFM calls

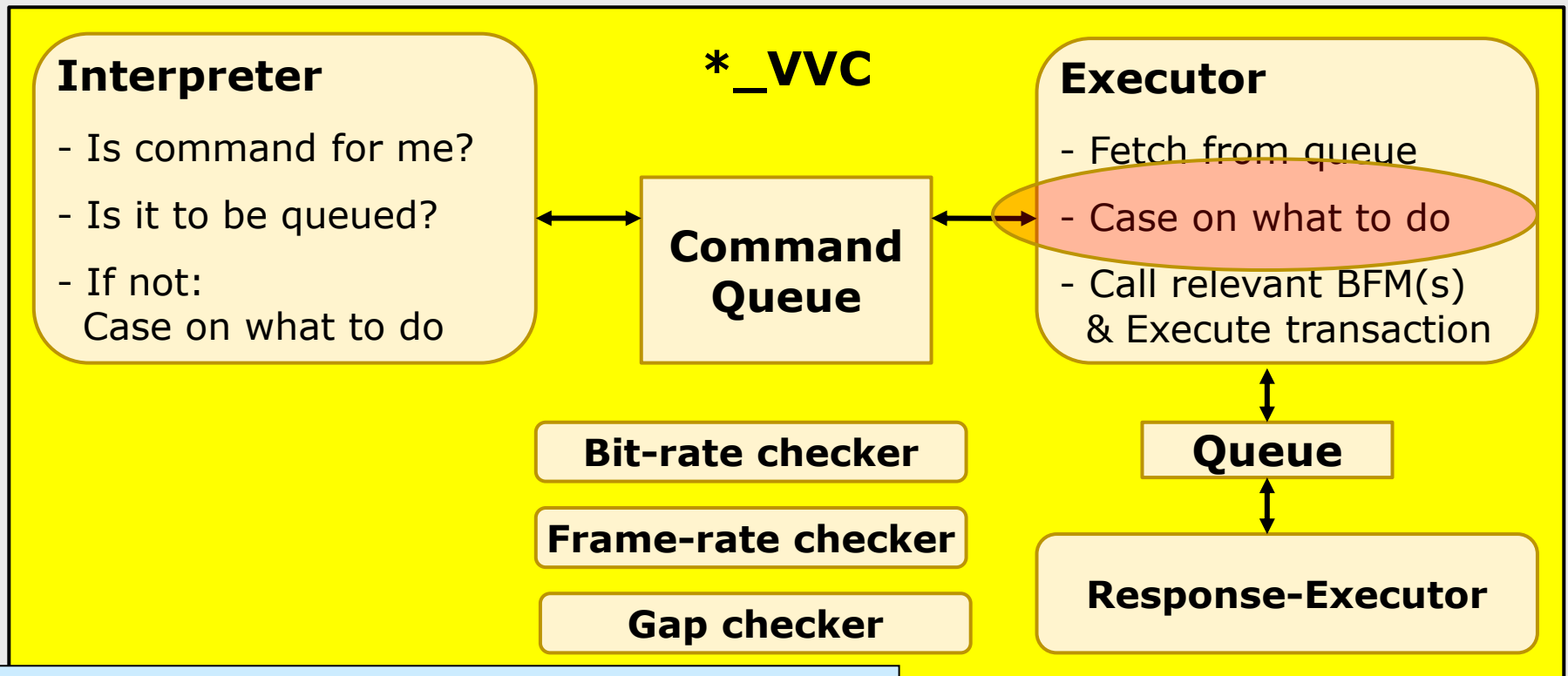
→ Standard VVC internal architecture

## VVC Generation

UART BFM to UART\_VVC:  
**less than 30 min**  
(using `vvc_generator.py`)

# VVC: Easy to extend

- Easy to handle split transactions
- Easy to add local sequencers
- Easy to add checkers/monitors/etc
- Easy to handle out of order execution



- Standard Queuing system
- Standard handling of multithreaded interfaces
- Standard control of parallel checkers

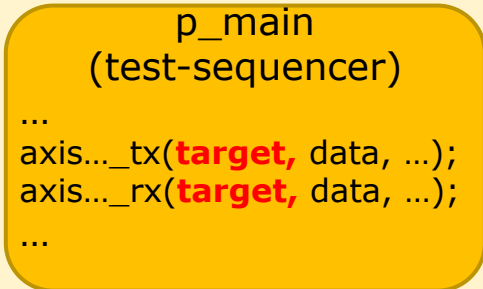
# VVC Advantages

- 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

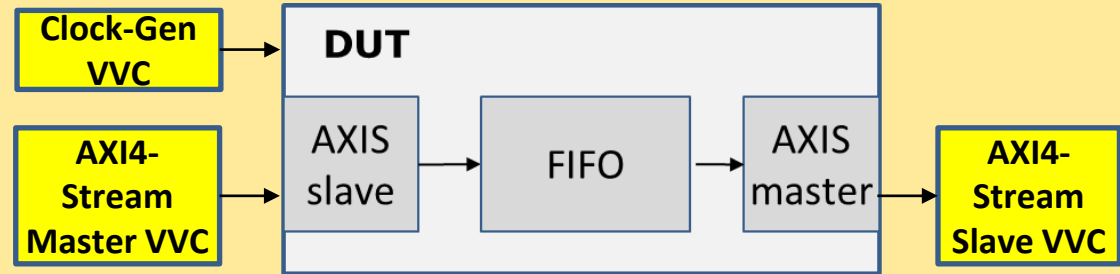
Better Overview, Maintenance, Extensibility and Reuse

# AXI-stream - VVC based TB (1)

## VVC based Testbench



## VVC based Test harness



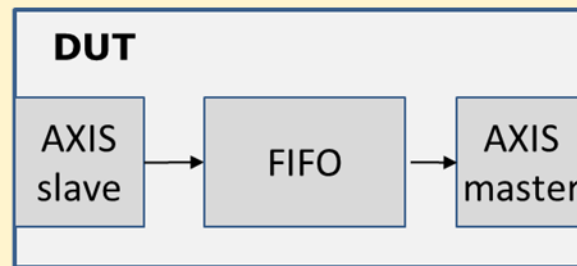
```
axistream_transmit(target, data, ...);  
axistream_expect(target, data, ...);
```

**clock\_generator**

**p\_main**  
(test-sequencer)

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

## BFM based Testbench

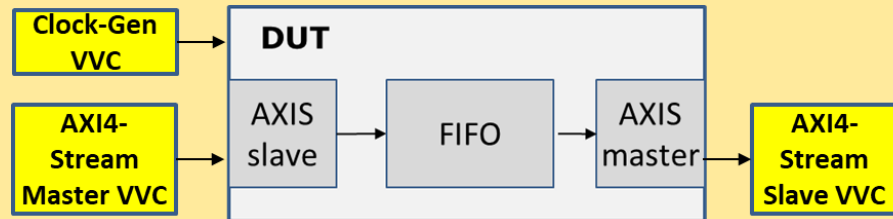


# AXI-stream - VVC based TB (2)

## VVC based Testbench

```
p_main
(test-sequencer)
...
axis..._tx(target, data, ...);
axis..._rx(target, data, ...);
...
```

## VVC based Test harness



## 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)
```

- 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

Generic to select Master or Slave

# Required code for AXI-Stream VVC

- Need to include core libraries and packages in code
- Need to include AXI-Stream library and packages in code:
- Define your AXI-stream record: (A must for pre-defined records)
- Define your signals and connect: (Mandatory in any case)
- Set tkeep = '1' to the slave BFM (Must indicate some way...)
- You are now ready to write any sequence of transmit, receive or expect:

```
library uvvm_util;  
context uvvm_util.uvvm_util_context;  
library uvvm_vvc_framework;  
use uvvm_vvc_framework.ti_vvc_framework_support_pkg.all;
```

```
library bitvis_vip_axistream;  
context bitvis_vip_axistream.vvc_context;
```

```
subtype t_axis is t_axistream_if(  
    tdata(7 downto 0), tkeep(0 downto 0),  
    tuser(0 downto 0), tstrb(0 downto 0),  
    tid(0 downto 0), tdest( 0 downto 0));
```

```
signal m_axis : t_axis;  
signal s_axis : t_axis;
```

```
s_axis.tkeep <= (others => '1');
```

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

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



# 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

# 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

m example

# Documentation VVC

## 1 VVC procedure details

Procedure	Description
<code>axistream_transmitf_bytes()</code>	<p><code>axistream_transmitf_bytes()</code> (VVCT, <code>vvc_instance_idx</code>, <code>data_array</code>, <code>luser_array</code>, <code>lstrb_array</code>, <code>id_array</code>, <code>dest_array</code>], <code>msg</code>, <code>[scope]</code>)</p> <p>The <code>axistream_transmit()</code> VVC procedure adds a transmit command to the AXI4-Stream VVC execution commands have completed. When the command is scheduled to run, the executor calls the AXI4-Stream BFM QuickRef.</p> <p>The <code>axistream_transmit()</code> procedure can only be called when the AXI4-Stream VVC is instantiated with 'GC_MASTER_MODE' to true.</p> <p>Examples:</p>

- Syntax + Overloads
- Examples
- Explanations

## 3 VVC Configuration

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

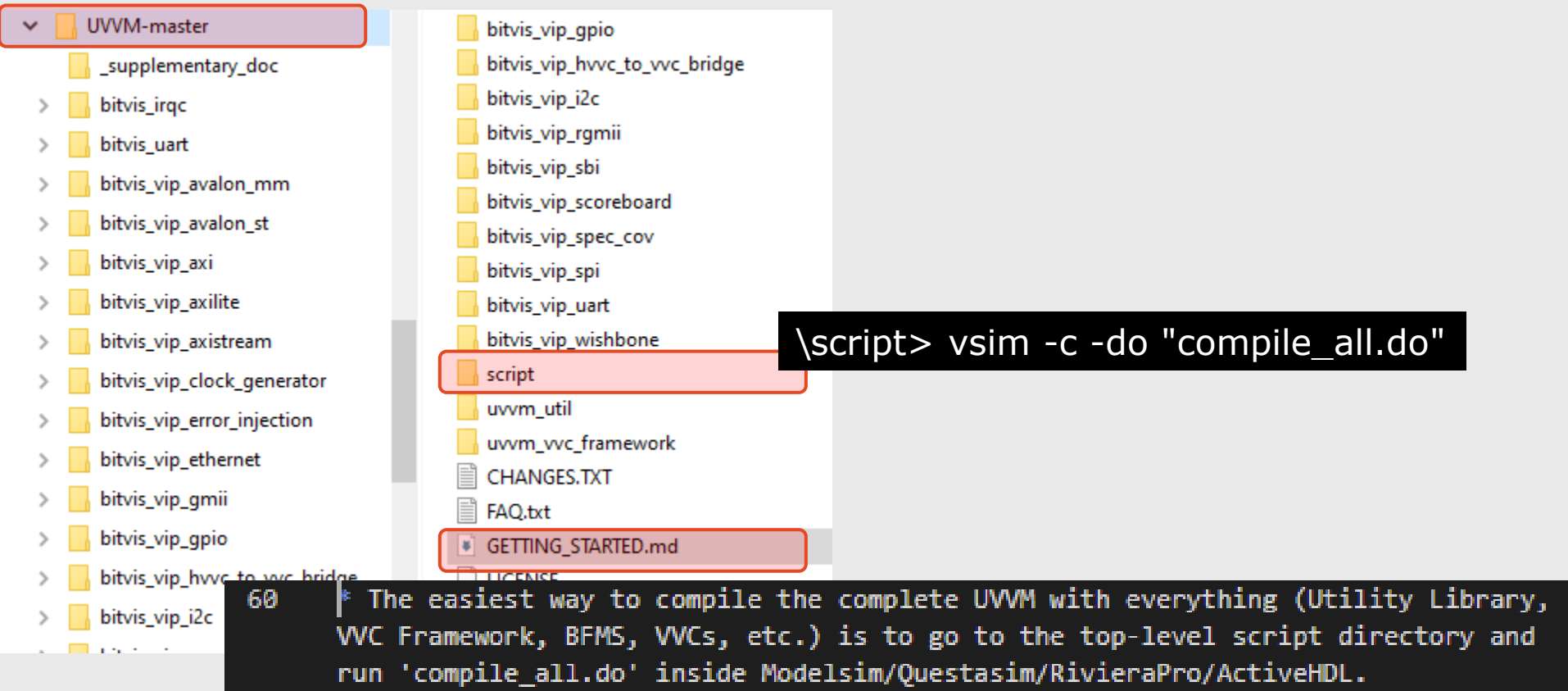
- BFM Config as for BFM
- Additional VVC setup

Defaults are fine...

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;
```

# Compiling UVVM



The screenshot displays a file explorer window showing the directory structure of the UVVM-master project. The 'script' directory is highlighted with a red box. A terminal window overlay shows the command `\script> vsim -c -do "compile_all.do"`. A text box explains the compilation process:

60 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.

# Advanced VVC usage

- May utilise more of the protocol – as for BFM
- May define different widths – as for BFM
- May configure behaviour – as for BFM
  - E.g. to set ready low duration to random : **(Same syntax for all VVCs)**

```
shared_axistream_vvc_config(1).bfm_config.ready_low_duration := C_RANDOM;
```

```
shared_axistream_vvc_config(1).bfm_config.ready_low_duration := C_RANDOM;
```

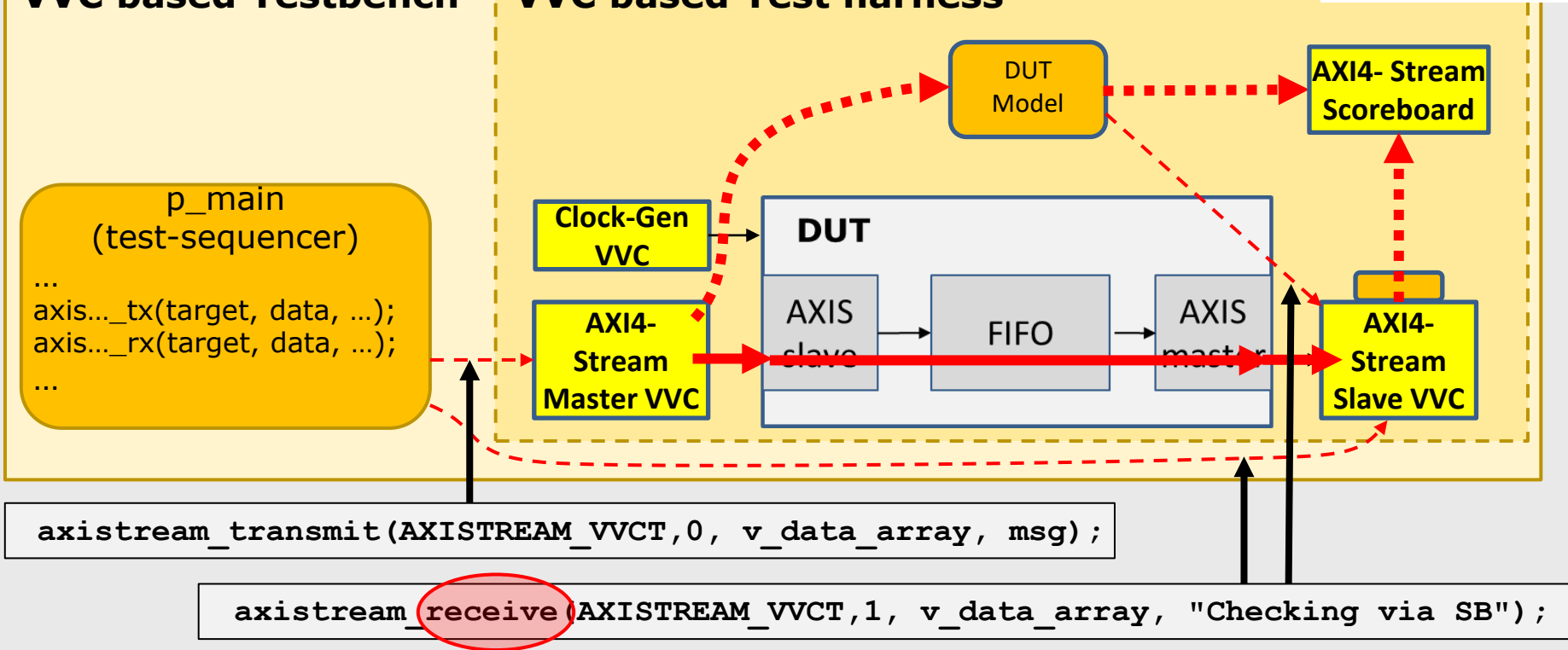
- Additional VVC features
  - Parallel stimuli/checks of multiple interfaces
    - ◆ All controlled from a single sequencer (or more if wanted)
  - Queuing of commands separately on each interface
  - Delay insertion to allow skewing of interface accesses
  - Transaction info available for advanced TBs
  - Activity watchdog
  - Etc...

# Advanced scoreboard-based TB



## VVC based Testbench

## VVC based Test harness



# Lot's of free UVVM BFM's and VVCs

Similar to the BFM's and VVCs for AXI-stream:

- AXI4-lite
- AXI4 Full
- AXI-Stream Master + Slave
- UART Transmit and Receive
- SBI
- SPI Master and Slave
- I2C Master and Slave
- GPIO
- Avalon MM
- Avalon Stream Master and Slave
- 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  
Free & Open Source  
VHDL Interface Models**

# The newer stuff

- ESA Extensions in ESA-UVVM-1
  - Scoreboarding
  - 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)
- Other general improvements
  - All Testbenches and Documentation sources made available
  - Lots of new and improved functionality in UVVM, BFM's and VVCs
  - New VVCS: Full AXI, Wishbone, GMII, RGMII, Ethernet
- Significant extensions coming in Q3 and Q4



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





# Pick and choose

- 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 - with any cmd

`AXI4-lite``GPIO``SBI``SPI``UART``GMII``RGMI``AVALON``AXI4-stream``I2C``AVALON stream``*_write()``*_check()``*_transmit()``*_receive()`

# Courses

- **Advanced VHDL Verification – Made simple**
  - Munich 26-28 October
- **Accelerating FPGA and Digital ASIC Design**
  - Munich 10-11 November
- More courses on demand/request
  - On-site, online, public. In Europe and outside Europe
  - May adapt or combine courses to your needs

## 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

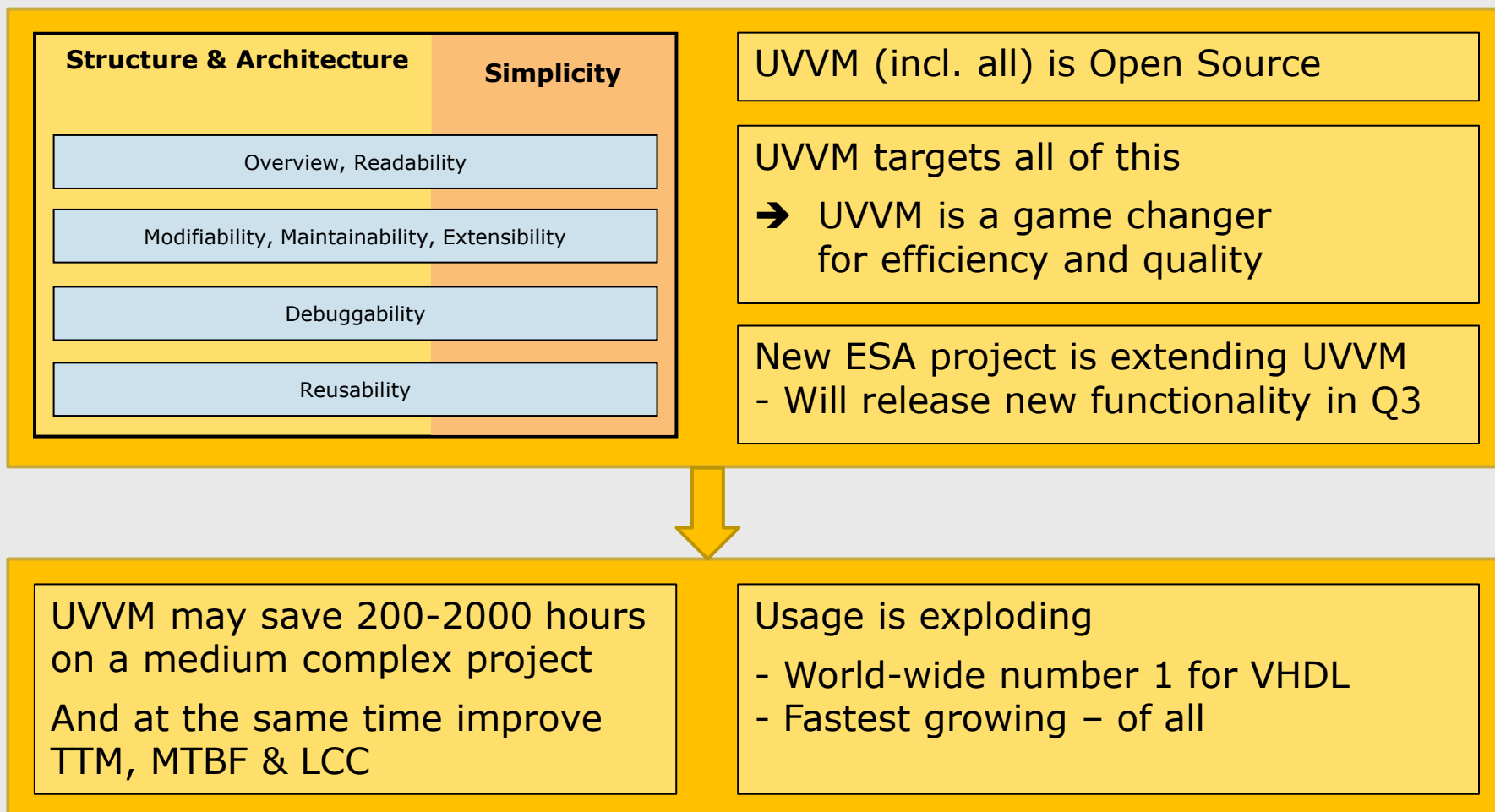
## Verification

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

<https://emlogic.no/courses/>

# Summary

- Huge improvement potential for more structured FPGA verification





# EmLogic

## Thanks for your attention

Community contributions to UVVM are very welcome...

Please let me know if this would be possible

[et@emlogic.no](mailto:et@emlogic.no)