EmLogic

UVVM
– An introduction to the world's #1 VHDL verification methodology

FPGA Conference Europe, 5 July 2022

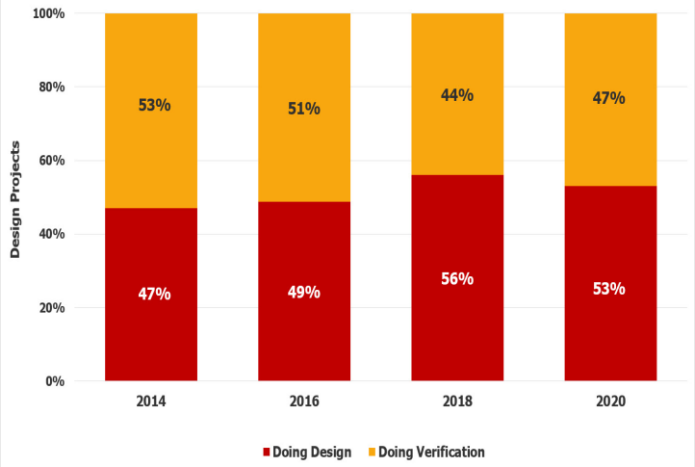
EmLogic

- Independent Design Centre for Embedded Systems and FPGA
- Established 1st of January 2021. **Extreme ramp up**
  - January 2021: 1 person
  - July 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

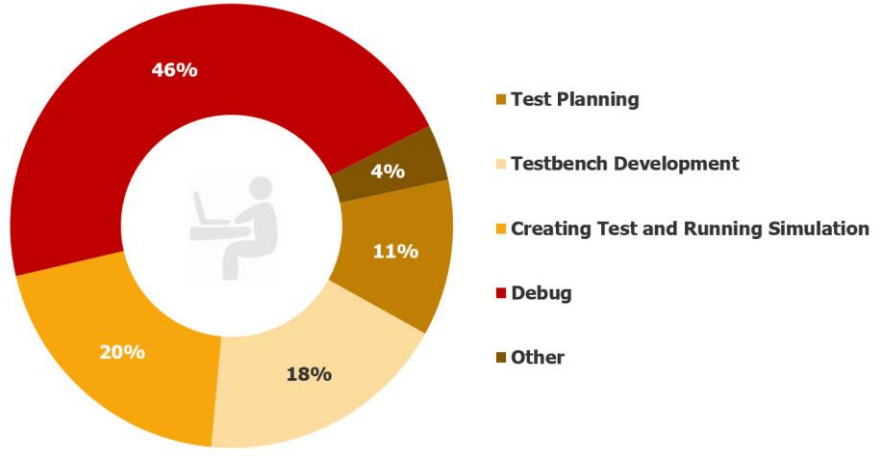
"Be afraid, be very afraid"

- *'The Design Warrior's Guide to FPGAs'* *(by Clive "Max" Maxfield)* **:**
  - *"SW designers look at other code with horror"*
  - *"RTL sets new standards for awfulness"*
  - *"The majority of designs are almost unintelligible to another designer"*
  - My experience: RTL is heaven compared to testbenches…

Half the development time on Verification

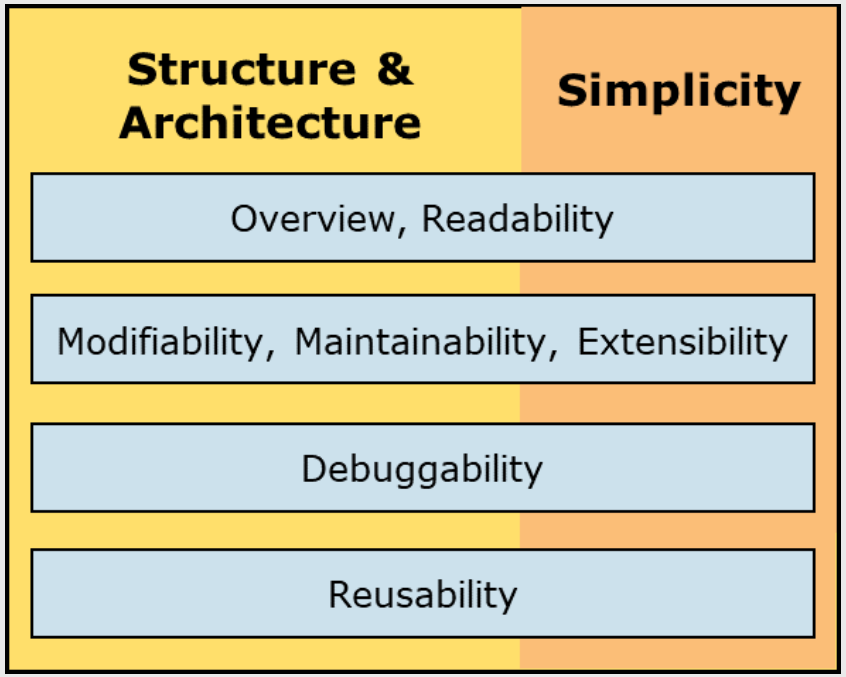


Nearly half the verification time on Debugging



EmLogic

Quality and Efficiency



Solution:

Architecture

Simplicity - where needed the most

Major Challenge:

Awareness

➔ Prioritise the reader at all times

UVVM targets all of this

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

The main architectural needs for TBs

- Depending on DUT complexity
  there are various architecture options, needs and levels
- But- there are mainly two **main** architectural approaches

1. No need to handle simultaneous activity on multiple interfaces
   - For DUT with no contention issues, no cycle related corner cases, etc

   ➔ Need only a simple TB with a single test sequencer process
   (using procedures and functions as needed, called from this process)

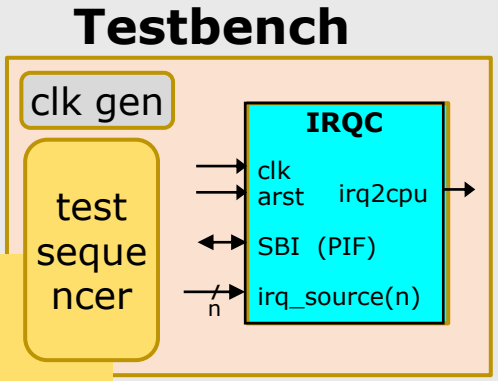
   - Including simple extension on this with no complex cycle related corner cases
     - E.g. Additional processes to apply and fetch data for a simple data flow DUT

2. Need to control and/or check multiple interfaces simultaneously
   - DUT has potential cycle related corner cases that need to be checked

   ➔ Need to run multiple threads (entities and/or processes) simultaneously

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, '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

UVVM Utility Library for simple **and** advanced testbenches

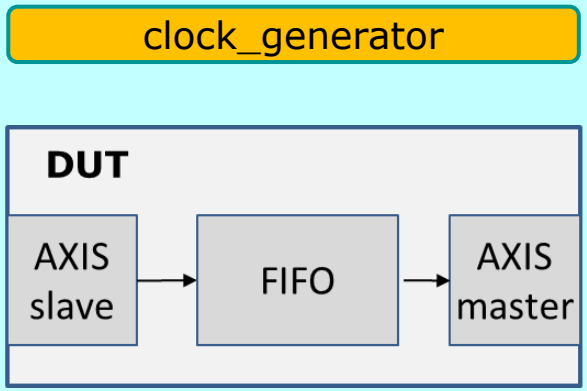
- check_stable(), await_stable()
- clock_generator(), adjustable_clock_generator()
- random(), randomize()
- gen_pulse()
- block_flag(), unblock_flag(), await_unblock_flag()
- await_barrier()
- enable_log_msg(), disable_log_msg()
- to_string(), fill_string(), to_upper(), replace(), etc…
- normalize_and_check()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

AXI-stream - BFM based TB

BFM based Testbench

p_main (test-sequencer)

```
...
axistream_transmit(data, ...)
axistream_receive(data,...)
axistream_expect(data, ...)
...
```



- No test harness (for simplicity)
- Sequencer has direct access to DUT signals
  - Thus BFMs from p_main can also see the DUT signals
- BFMs are sequential procedures running in sequence in p_main.

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

Advanced BFM usage - in simple TB

```
tkeep, tuser, tlast,
tstrb, tid, tdest
```

- 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…

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`	

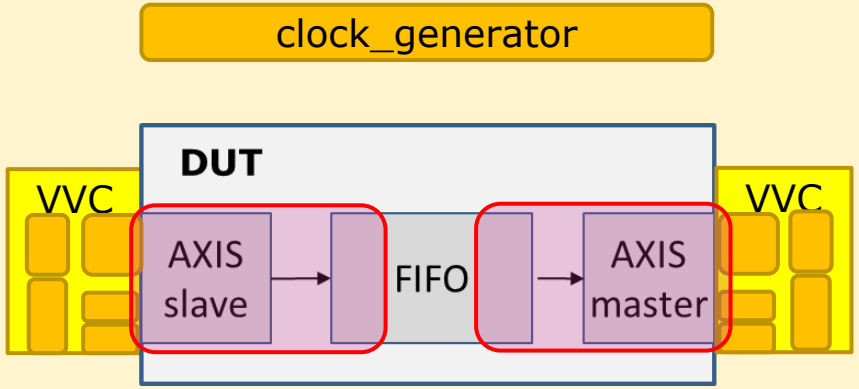
EmLogic

What if we need to check the DUT for simultaneous activity?

BFM based Testbench

p_main (test-sequencer)

```
...
axistream_transmit(data, ...)
axistream_receive(data,...)
axistream_expect(data, ...)
...
```



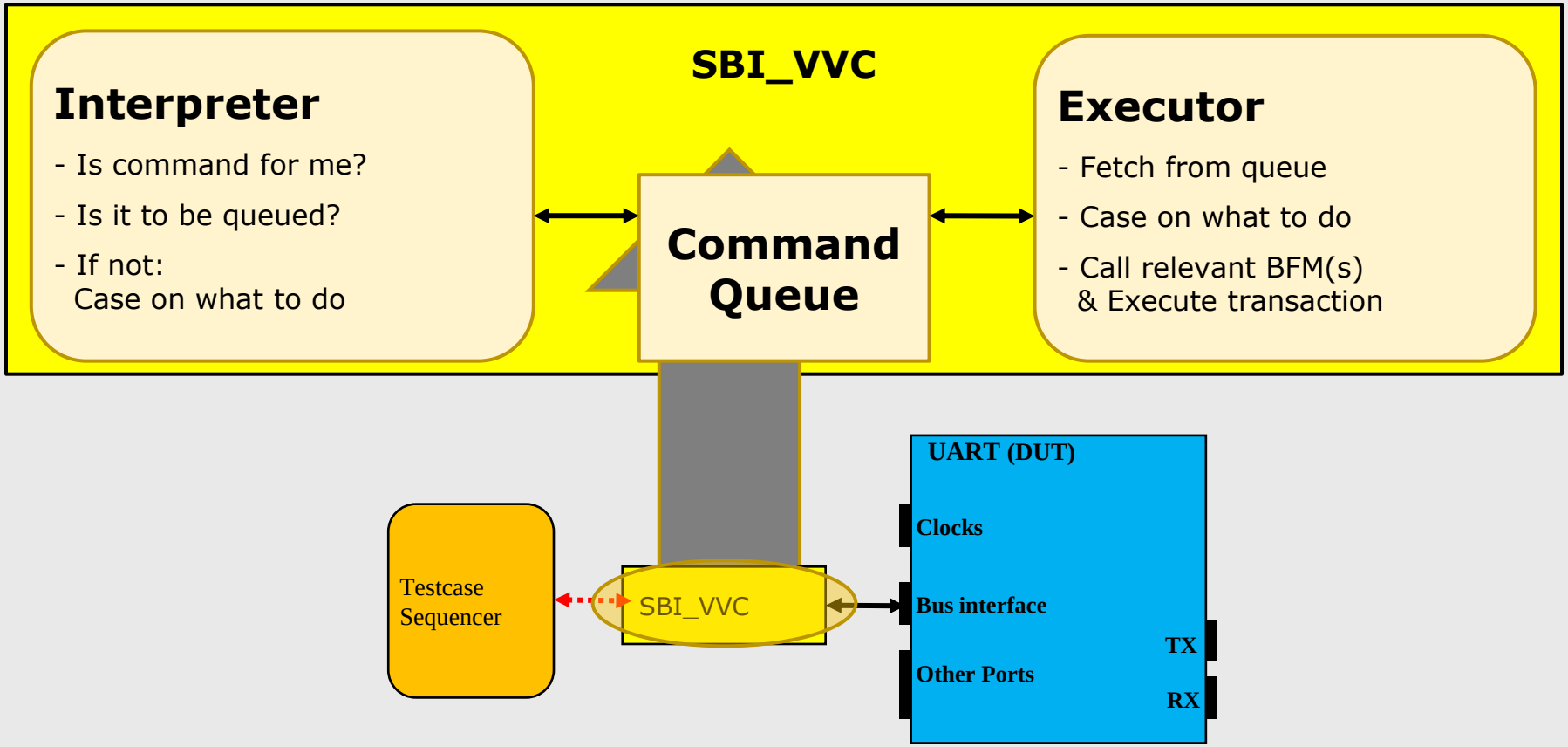
- BFMs are great for simple testbenches
  - Dedicated procedures in a simple package
  - Just reference and call from a process
- BUT
  - A process can only do one thing at a time
    - Either execute that BFM
    - **Or** execute another BFM
    - **Or** do something else

- To do more than one thing:
  - → Need multiple "threads"
  - → Could use multiple processes
    - Need inter process communication
  - → Leads to chaos – as for design

→ Need an entity (or component)
(VC = Verification Component)

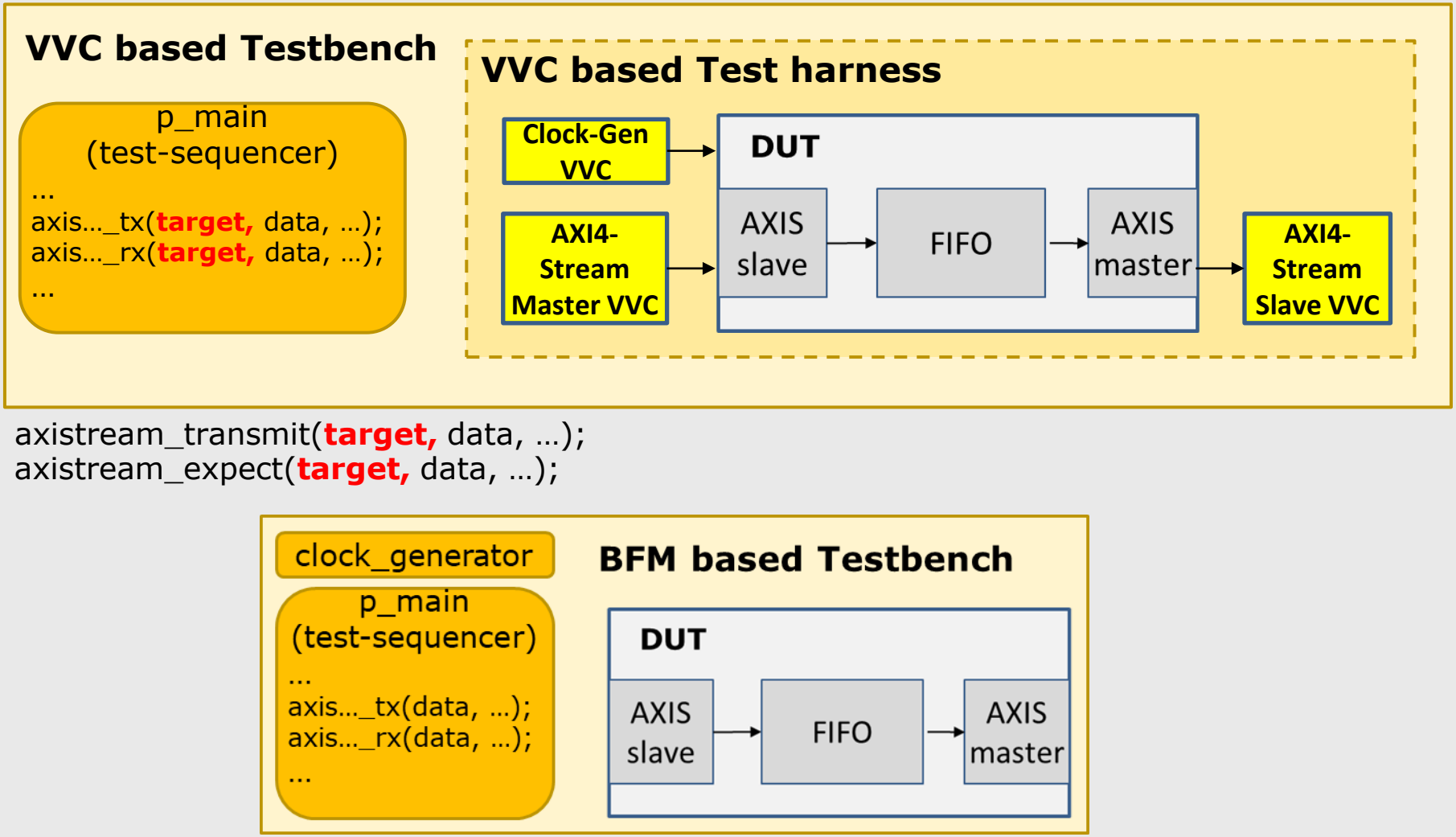
→ Need a defined protocol

VVC: VHDL Verification Component



EmLogic

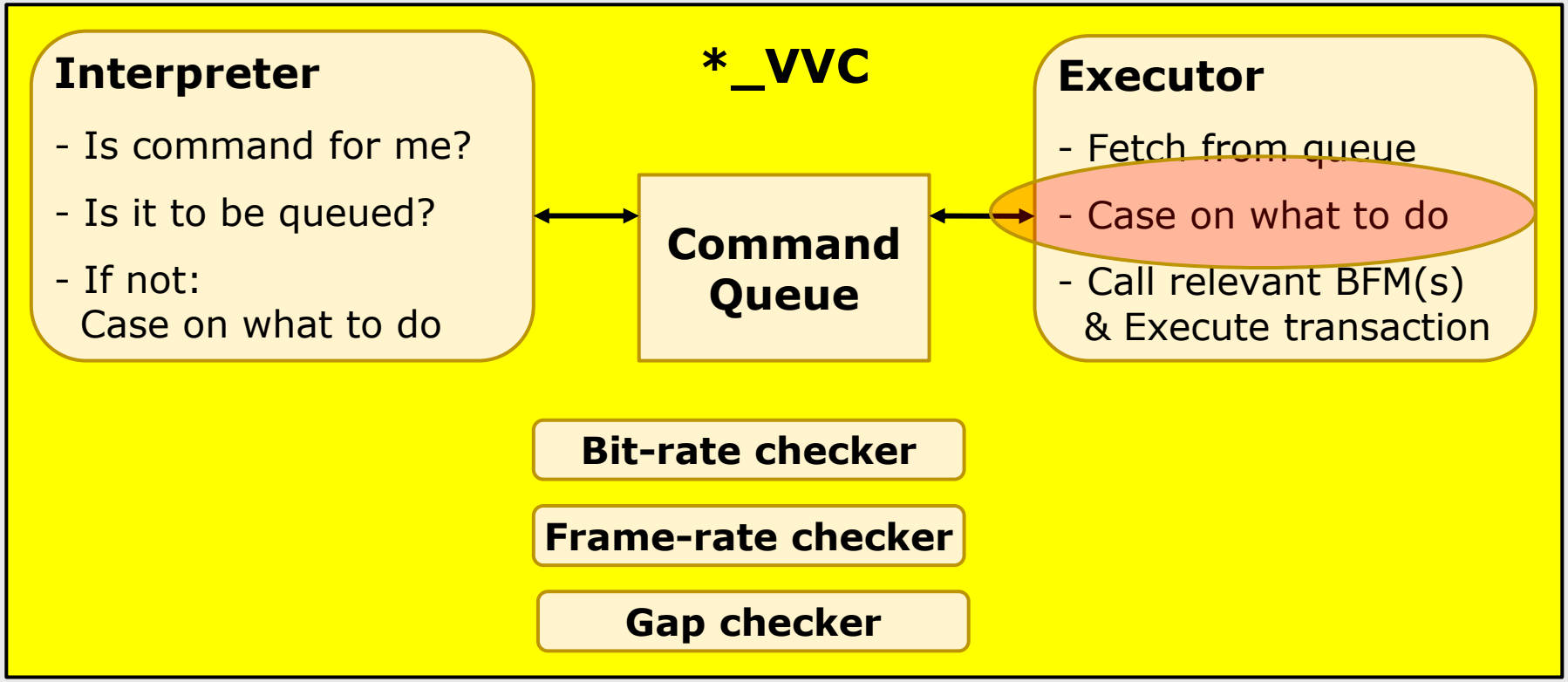
AXI-stream - VVC based TB (1)



VVC: Easy to extend & modify

- Easy to add local sequencers

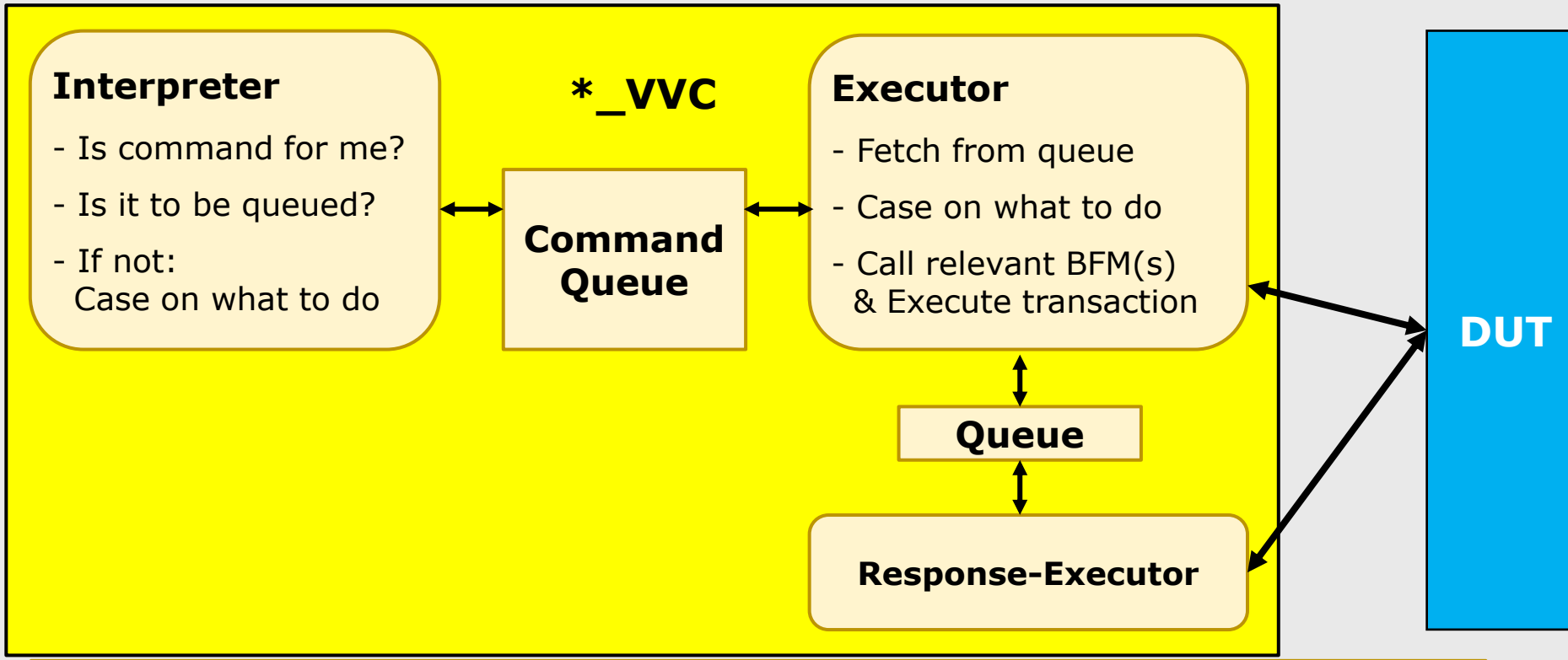
- Easy to add checkers/monitors/etc



Checkers are better included as parallel processes.
VVC architecture allows simple inclusion and control

VVCs: Extended

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



Multi-thread interface handling is better controlled as a pipeline
VVC architecture allows simple inclusion – with queuing and control

➔ Allows overview to be kept – for something that normally creates chaos…

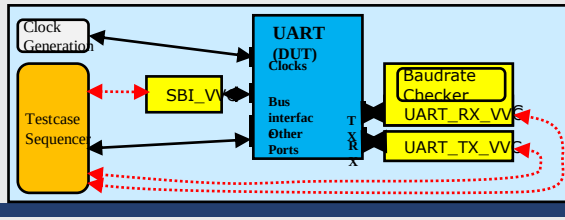
VVC Advantages

Due to:
- VVC architecture
- TB architecture
- Command structure

- 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

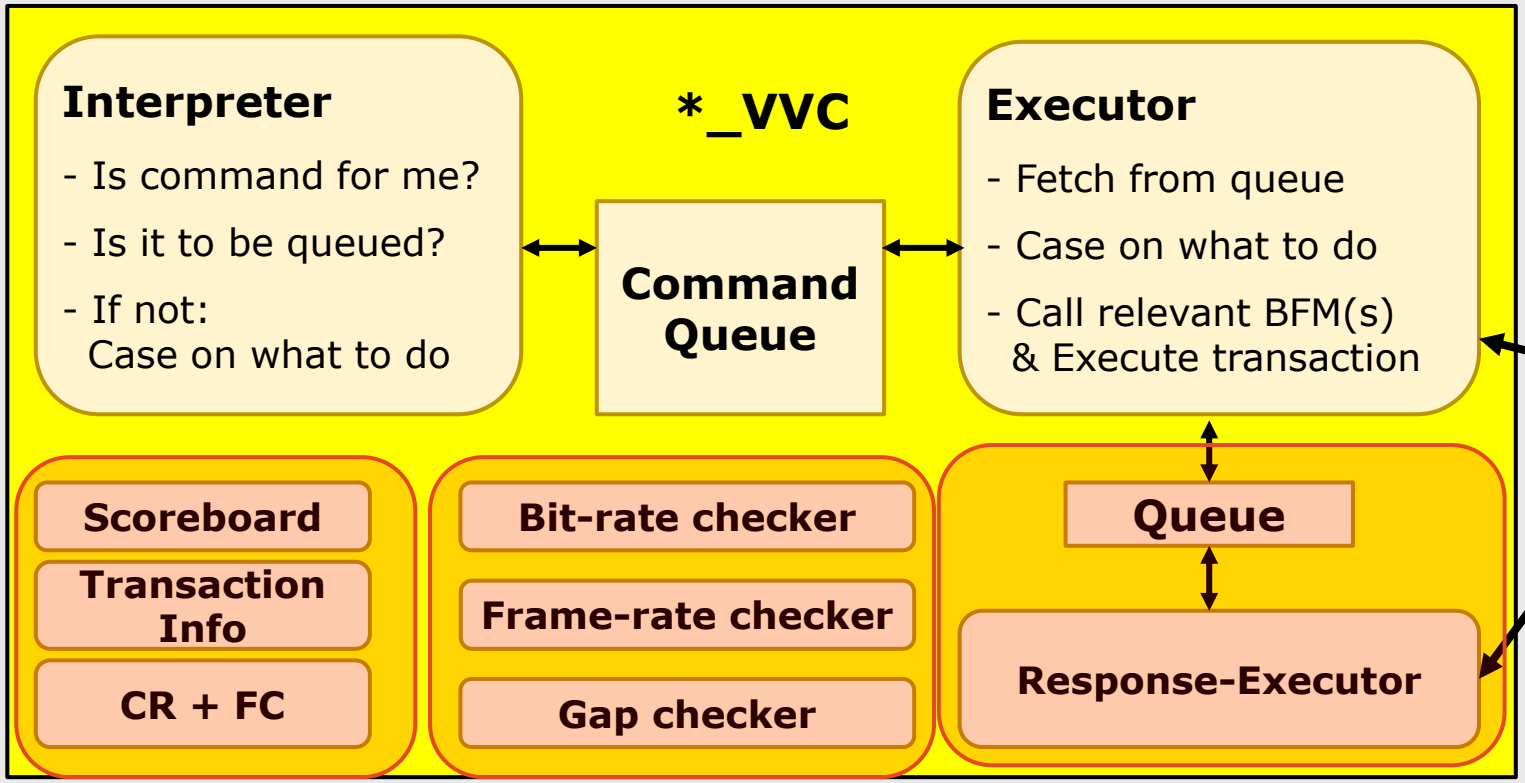
Better Overview, Maintenance, Extensibility and Reuse

VVCs: Too advanced???



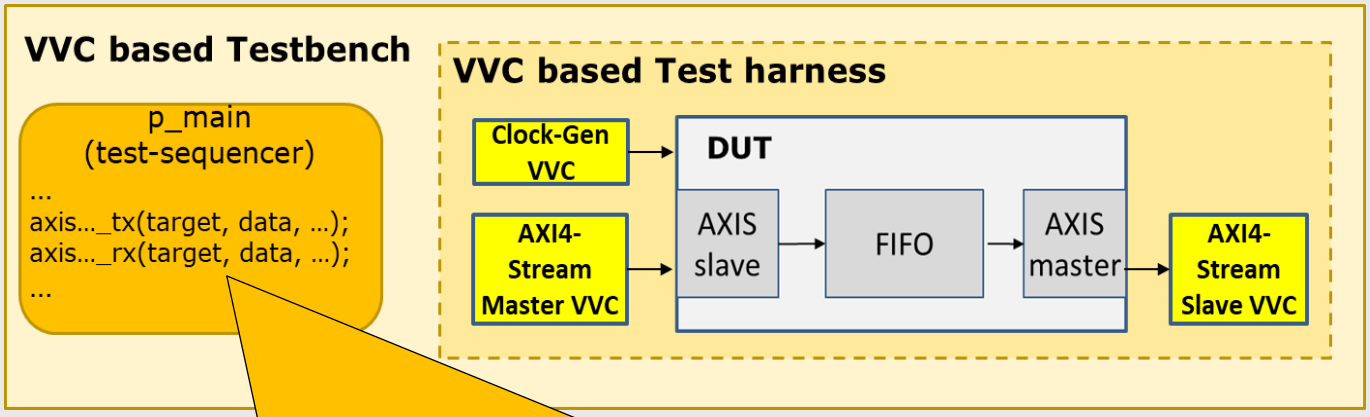
Advanced functionality is great when needed, but what if not???

- If using an existing VVC, just ignore it. Use it out of the box – without the extras.
- If making your own VVC, don't include the advanced stuff
  Skip it in the VVC generator, or don't include it if you copy the architecture.



EmLogic

Simplicity where needed the most



TB with harness

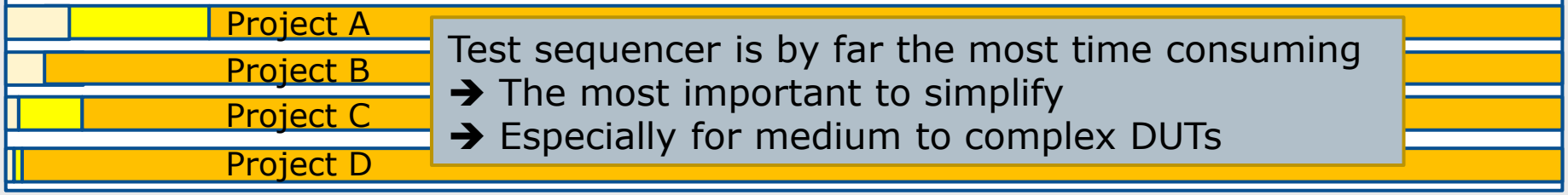
VVCs

Test sequencer

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

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

Totalt Workload percentages for various scenarios



Test sequencer is by far the most time consuming
➔ The most important to simplify
➔ Especially for medium to complex DUTs

Not investing in a good architecture ➔ Test sequencer time may increase a lot

EmLogic

Lots 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

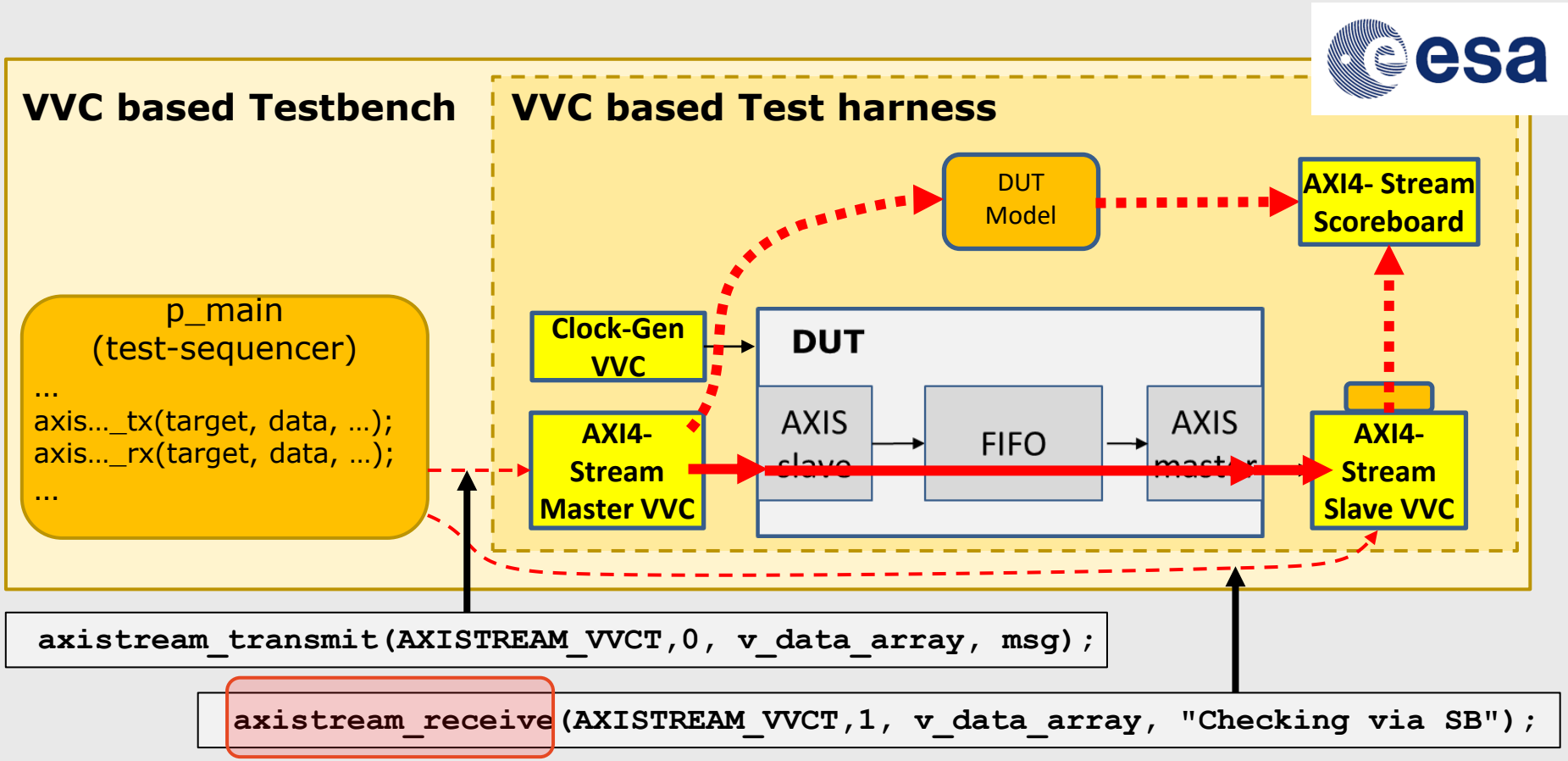
VVC: VHDL Verif. Comps.

- 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

Advanced scoreboard-based TB



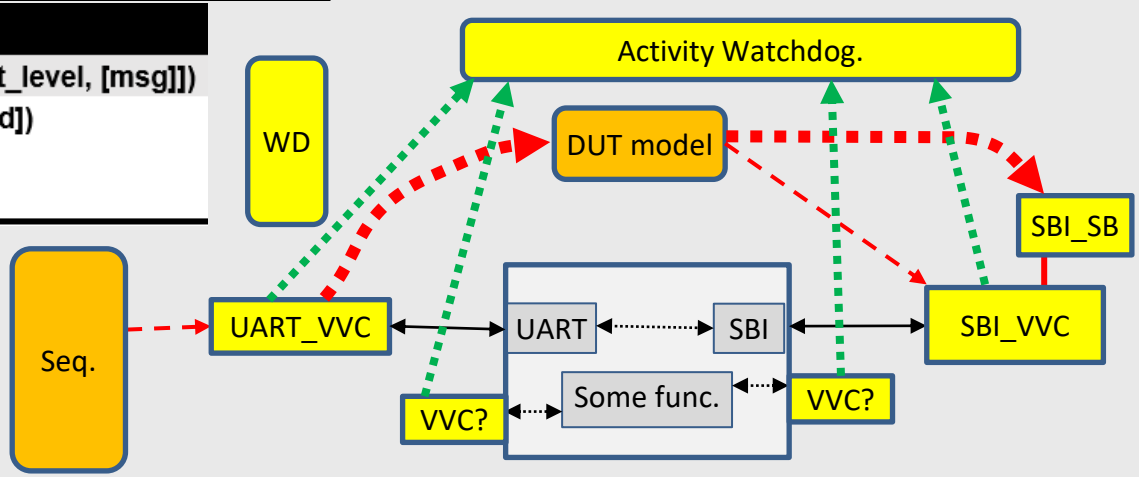
Model and Scoreboard allow functionality to be removed from test sequencer.

➔ Simplifies both overview and test sequencer

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)



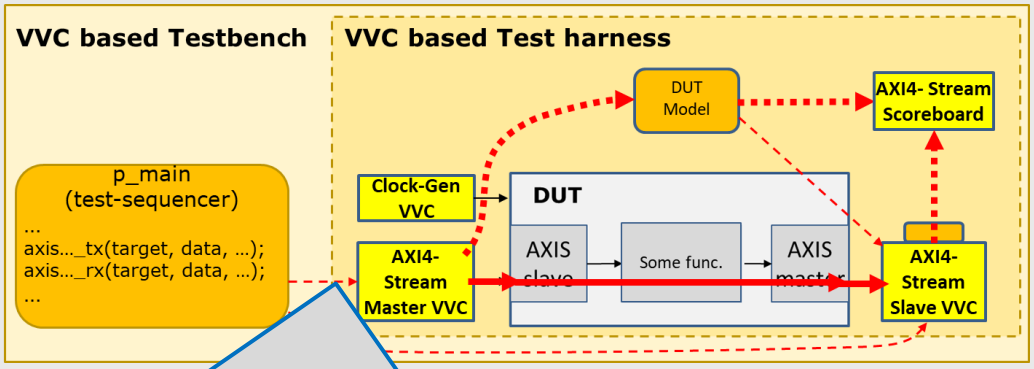
Activity WD VVCs and UVVM

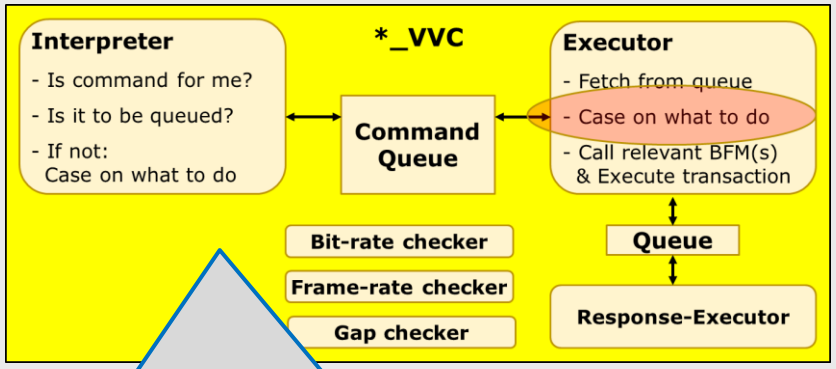
```
activity_watchdog(timeout, num_exp_vvc);
```

Apply both concurrently

Watchdogs allow a simpler test sequencer and better overview

A good architecture allows better understanding at all levels





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

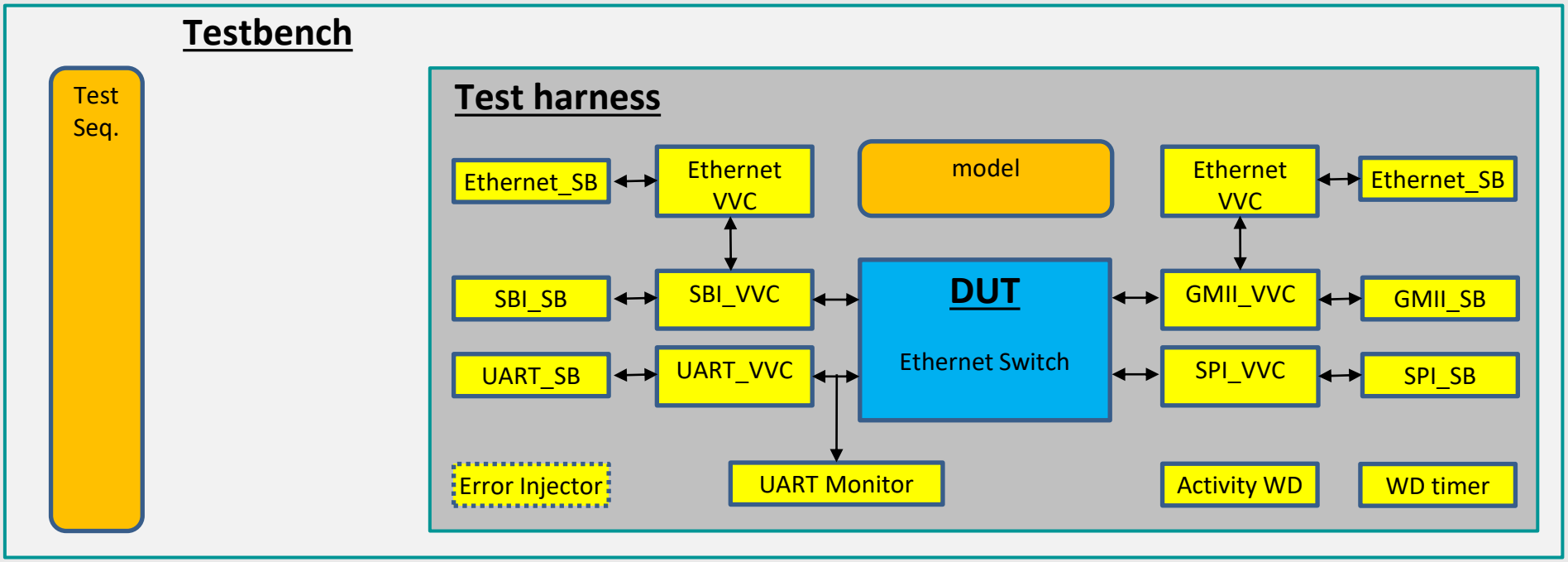
- Standard VVC internal architecture
- Standard VVC control of checkers
- Standard queuing system
- Standard handling of multi-threaded interfaces
- Standard debug support

Simplification **l for** **VVCs from different users will work together**

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

The full picture (1)

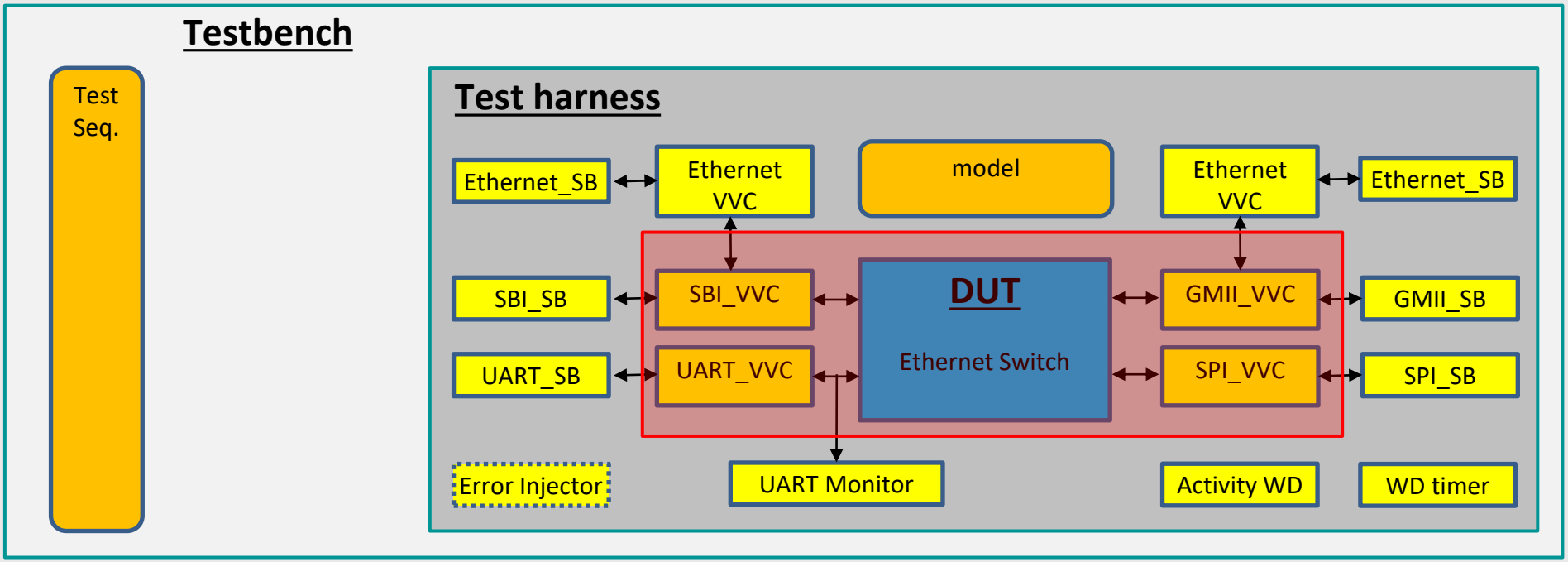
Is this too much? Too many VIP components? Too much structure?



EmLogic

The full picture (2)

VVCs are needed to check multiple interfaces simultaneously



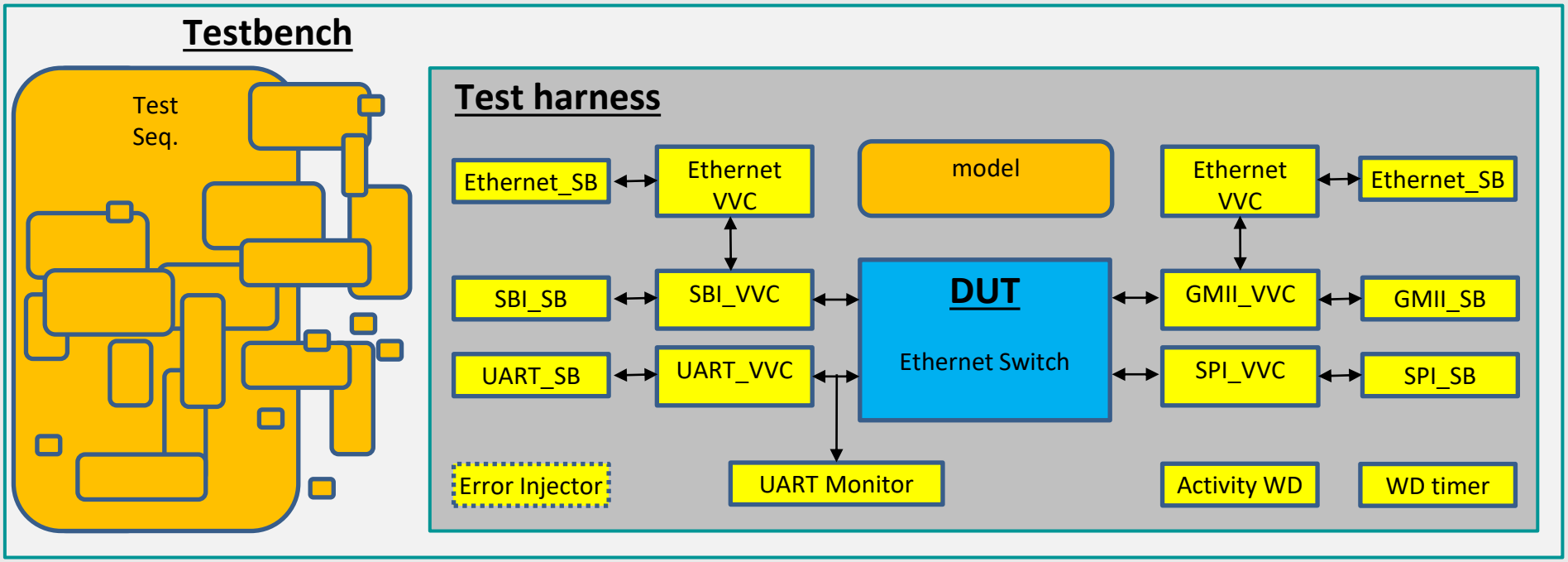
EmLogic

The full picture (3)

Removing VIPs ➔

Putting What if we simply “test” this?? sequencer

- only very unstructured, less overview, not maintainable, not very reusable



Debugging is much easier with a good architecture

(25% of development, acc. to Wilson Research 2020 survey)

EmLogic

Testbench examples?

- Available as example testbenches:
  - Demo TBs - from main **UVVM** repo on Github:
    - UART, Ethernet, IRQC, Scoreboard, Error injector, Spec_cov
  - Maintenance TBs - from **UVVM_Supplementary** on Github:
    - All VIPs
- May use examples as starting point for your TBs

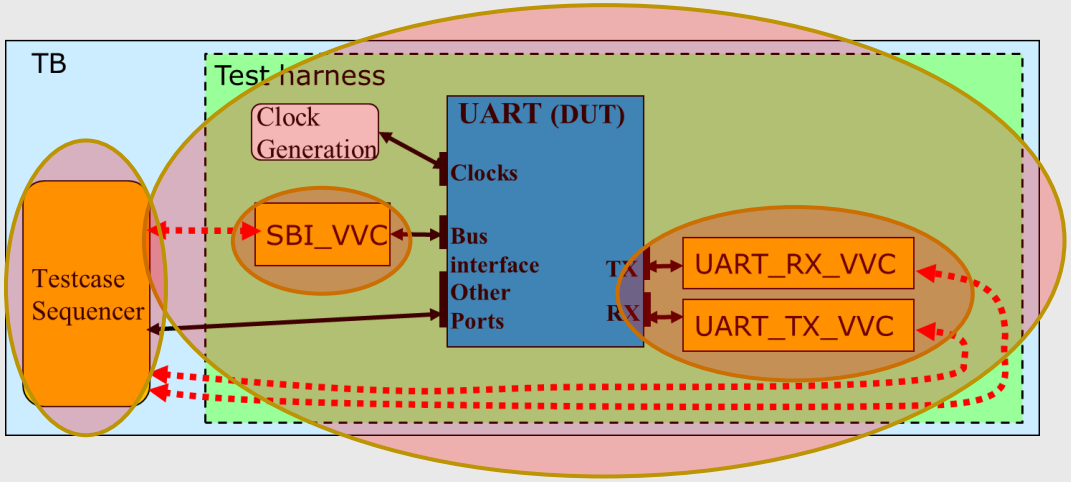
The three main development areas for adv. TBs
vs structure and efficiency evaluations

The central sequencer

- Always by far the most time consuming
- Massively simplified (cmd + sync)
- Even a SW designer can read it – **and** write it
- Any number of VVCs easily controlled
- Huge time saving where it matters the most

Test harness

- Dead simple
- Anyone can understand it
- Anyone can understand the interaction



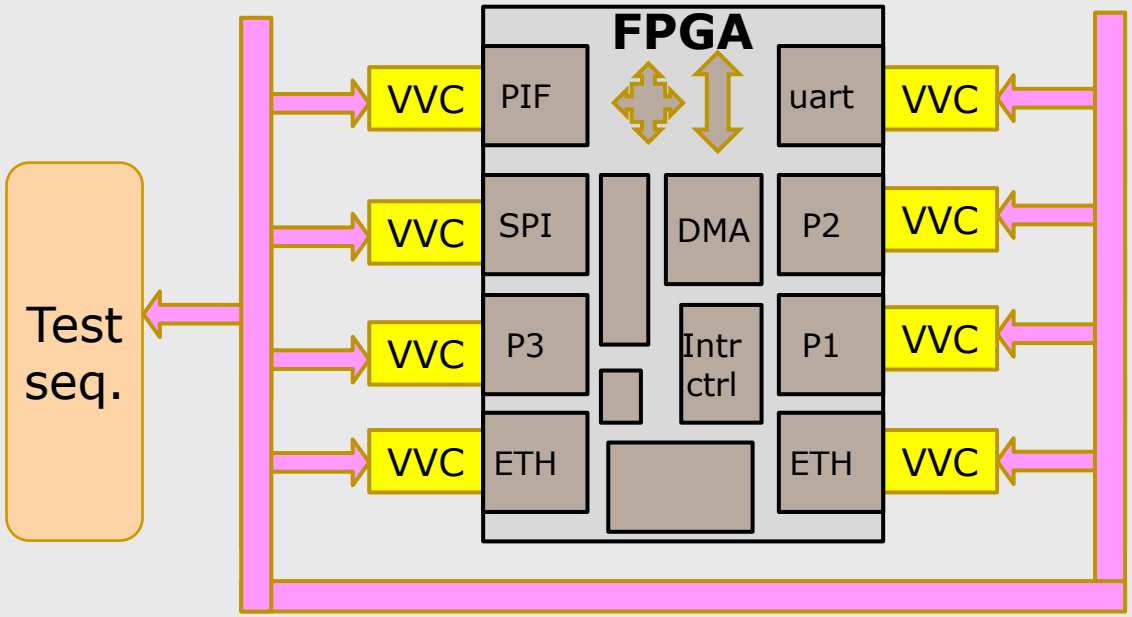
Verification Components

- More complex than just a BFM or a simple model (Unit with BFM)
- Far more functionality (common cmds +q +synchronization +multi/broadcast +skewing +trans.info +checkers ++++)
- Simplifies complex protocols significantly (BFM++)
- Allows really simple test harness
- Yields a huge improvement for testcase writers
- Significantly saves total verif time
- 30 min from BFM to VVC!!!

EmLogic

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 (process) of sequential commands only



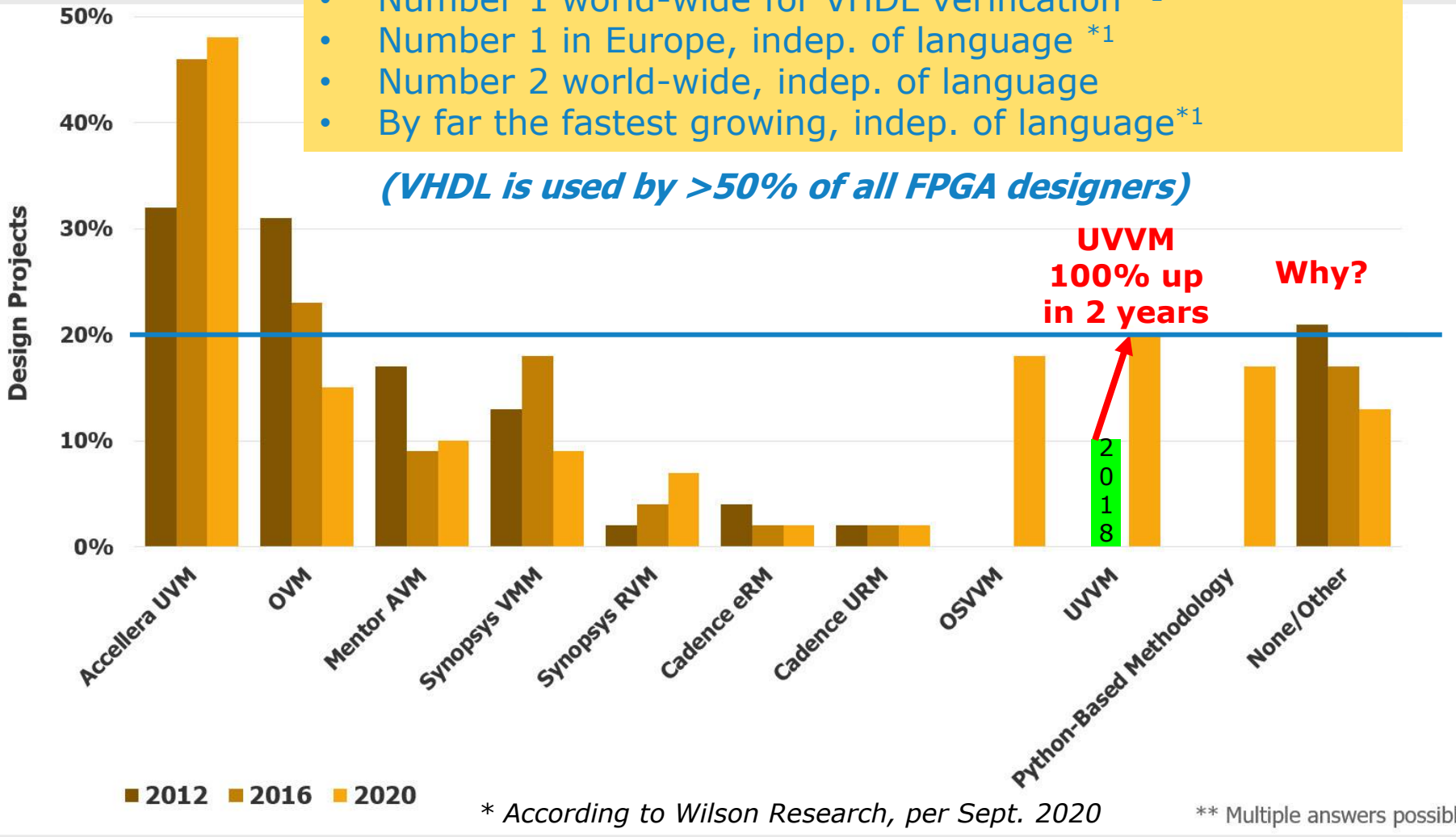
EmLogic

UVVM – World-wide #1

FPGA Verification Methodologies, world-wide, all languages

- 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

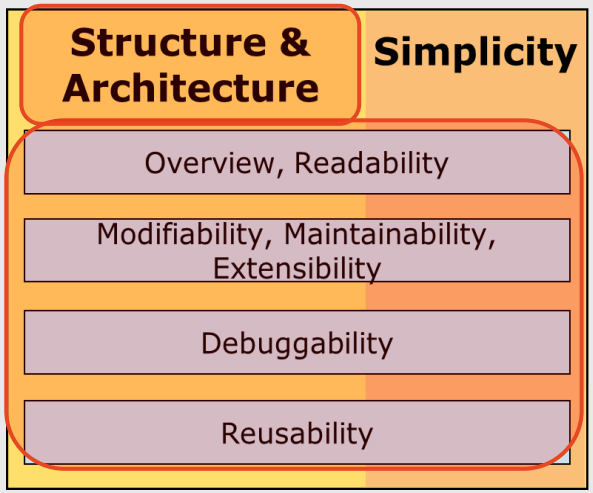
(VHDL is used by >50% of all FPGA designers)

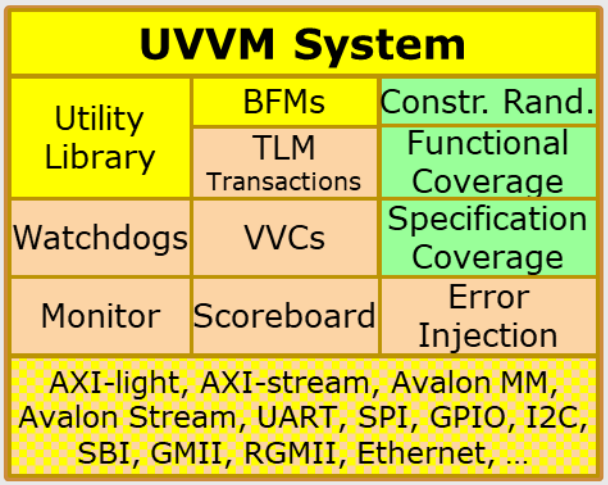


* *According to Wilson Research, per Sept. 2020*

** Multiple answers possible

UVVM enables Quality and Efficiency





Huge improvement potential in most projects	
Save **100-1000 hours** in low-medium complexity projects Save **500-3000 hours** in medium to high complexity projects	**+ TTM** **+ MTBF** **+ LCC**

Other presentations and tutorials

- 09:45 today: Presentation
  UVVM Enhanced Randomisation and Functional Coverage – and how this will help you make a better VHDL testbenches

- 09:00 – 10:30 tomorrow: Tutorial
  Making a structured VHDL testbench – for beginners

- 11:15 – 12:45 tomorrow: Tutorial
  Making an advanced testbench using models, scoreboards, verification components, high-level transactions and more

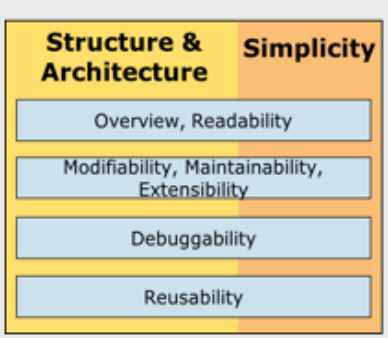
See also https://emlogic.no/courses/ for our courses

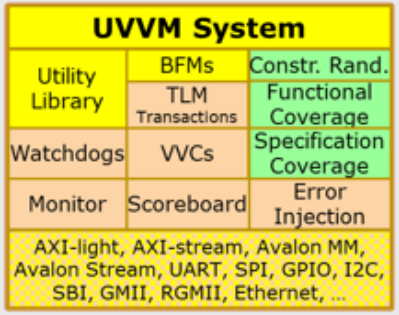
4-6 October, Germany, Frankfurt ? : 3-day VHDL Verification & UVVM

8-9 November, Germany, Frankfurt ? : 2-day FPGA and Digital ASIC Design

Thank you for attending

UVVM enables Quality and Efficiency





Huge improvement potential in most projects	
Save **100-1000 hours** in low-medium complexity projects Save **500-3000 hours** in medium to high complexity projects	**+ TTM** **+ MTBF** **+ LCC**

EmLogic