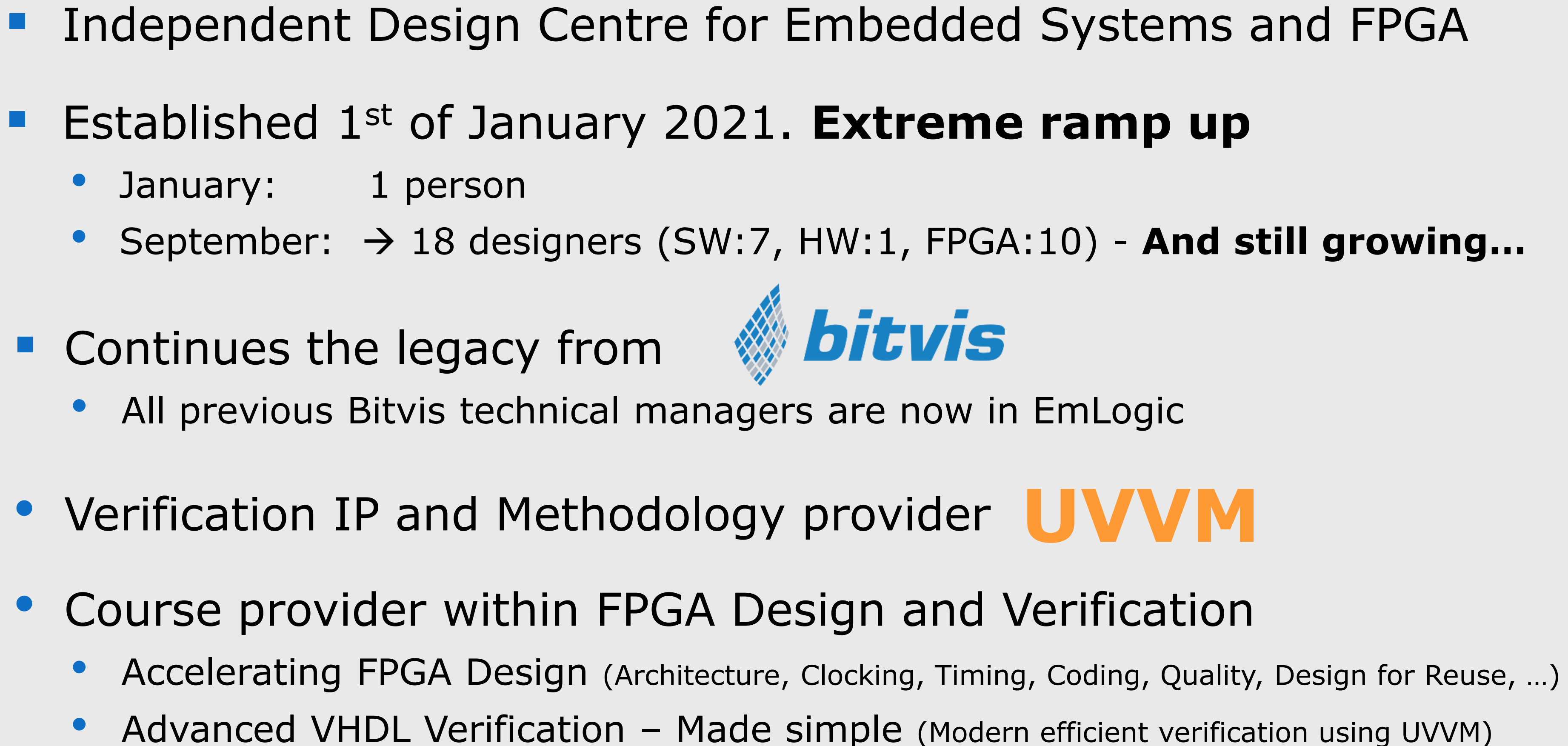
EmLogic

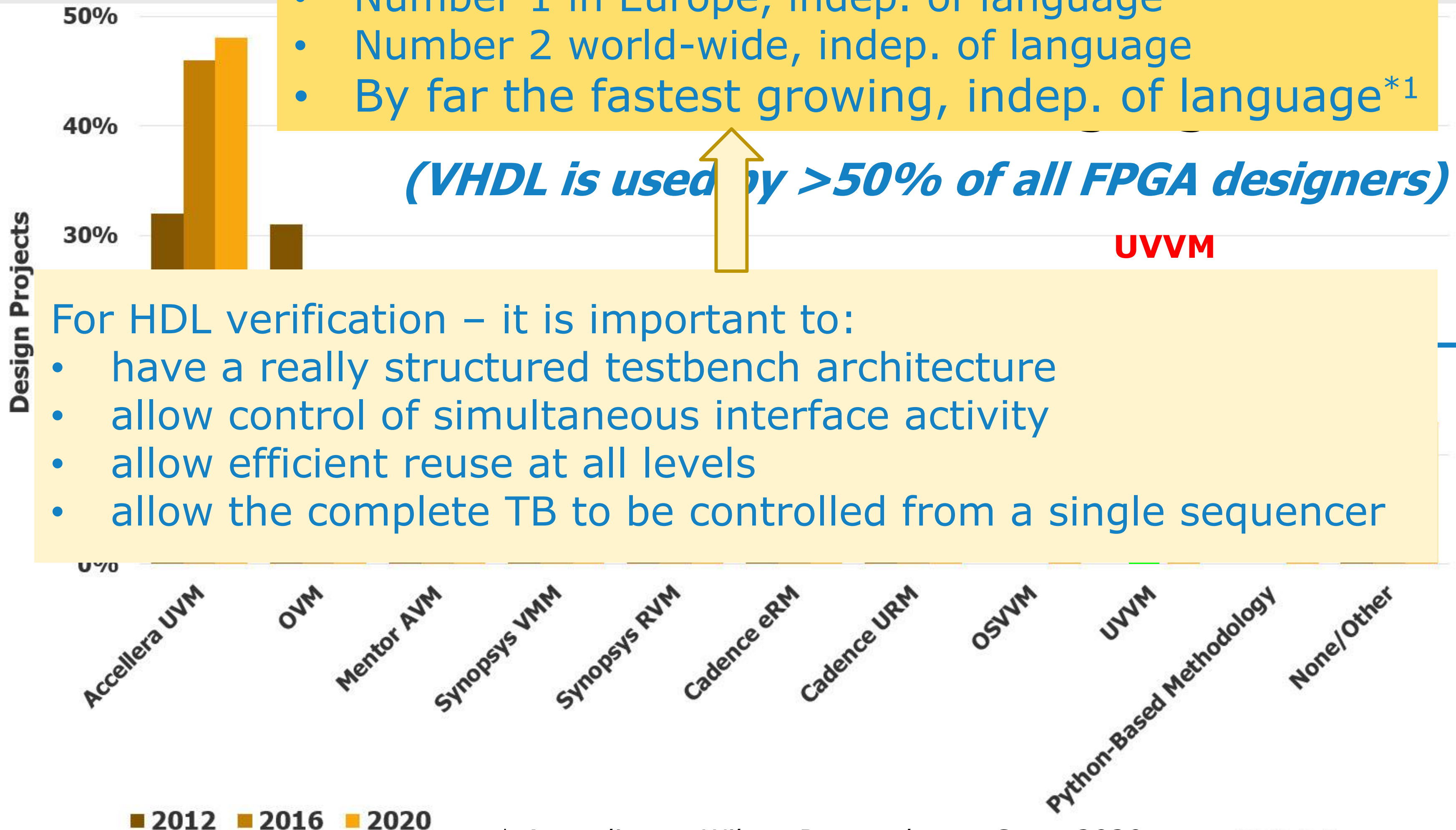
UVVM
- Brand new features
from the world's #1
VHDL Verification Methodology

FPGA Verification Day, Live Online, 23 September 2021

EmLogic

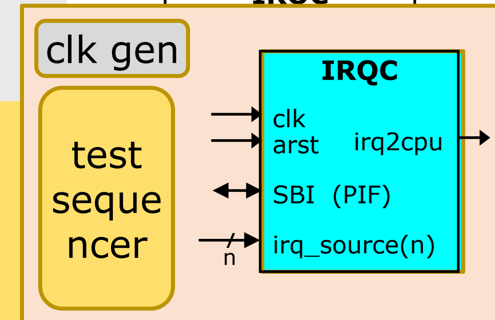


UVVM – World-wide #1



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);
```

All procedures with:

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

Lot’s of free UVVM BFMs and VVCs

- AXI4-lite
- AXI4 Full
- AXI-Stream 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

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

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)

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

- In addition lots of general improvement have been made

Today: Focus on New Features

- For an introduction to current functionality like:
  - More key functionality in Utility Library
  - BFMs : Functionality and usage
  - VVC and their benefits
  - The new functionality over the last 2 years
  - Why UVVM is #1

  Check out my previous Webinars, Posts, and Presentations

  The latest presentations being:

  MODERN VHDL TESTBENCHES
  AN AXI-STREAM EXAMPLE, FIRST dead simple, - THEN advanced - Both as simple as possible
  At FPGA Conference Europe, 6 July 2021. (Get in touch if you can't get it there)

  UVVM
  The main benefits of the world's #1 VHDL Verification Methodology
  At Mentor/Siemens Verification Webinar Series, 4 May 2021
  Complete presentation (webinar) available here:
  https://webinars.sw.siemens.com/uvvm-the-main-benefits-of-the/room

EmLogic

Brand New **1** - October 2021

- UVVM has had basic Randomisation since 2015
  - Good enough for **most** designers and verification engineers but….
  - We have got many requests for:
    - More advanced Randomisation in UVVM
    - Better integrated verification – than current alternatives
    - More understandable randomisation APIs
- UVVM now meets these requests with brand new:
  - Enhanced Randomisation
  - Optimised Randomisation

Basic Randomisation in "old" UVVM

- Under UVVM Utility library : methods_pkg
- Simple functions - using shared variable seeds:
  - my_int := random(VOID);
  - **my_int := random(4, 245);**
  - my_slv8 := random(8);
  - my_byte_array := random(1, 16);
  - my_time := random(1 ns, 15 ns);
- Also provides support for fixed random sequence
  - Needs to control seeds locally
  - random(4, 245, seed1, seed2, my_int);

Still the simplest solution for simple Randomisation

Enhanced Randomisation

- Located under UVVM Utility library : rand_pkg
  - New package included in context file
    → Will be available automatically once released
- Uses protected types

```
variable my_addr : t_rand;  -- The only preparation reqd.
```

- Allows far better control of randomisation – when needed
  - Combine ranges and set of values
  - Exclude set of values
  - Dedicated control of: 'with replacement' vs 'without replacement'
  - Additional multi-method approach for even more detailed control

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

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

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

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**

EmLogic

Other features

- General Initialisation

```
.set_rand_seeds(string|integers)   &   .get_rand_seeds()
```

```
.set_name("address generator")     &   .get_name()
```

```
.set_scope("UART TX")              &   .get_scope()
```

- General functionality configuration
  - Randomisation Distribution + Characteristics (Std. Deviation)
  - Weighting of ranges vs sets
- Special features
  - Unique values in vectors (64 values with unique values between 0 and 255)

```
payload <= my_data_vector.rand(64, 0,255, UNIQUE);
```

  - No repeating until all values have been used

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


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

Multi-method approach (2)

- Slightly more object oriented
  - Thus allows simpler build-up multiple constraints

Multi-method approach is better for:
- Reuse of constraints – **if needed**
- More partial or complex constraints – **if needed**
- Modification of constraints – **if needed**
- Future functionality extensions (UVVM or other)

Single-method approach is better if :
- Simple constraints, AND
- No need for future extensions in your current testbenches

One approach does not affect the other,
But would not recommend to mix in the same testbench.

May report configuration

```
my_addr.report_config(VOID);
```

```
# UVVM:  ==================================================
# UVVM:  ***  REPORT OF RANDOM GENERATOR CONFIGURATION ***
# UVVM:  ==================================================
# UVVM:      NAME                    : MY_ADDR
# UVVM:      SCOPE                   : AXI4_Master
# UVVM:      SEED 1                  : 1969513907
# UVVM:      SEED 2                  : 1510976018
# UVVM:      DISTRIBUTION            : UNIFORM
# UVVM:      WEIGHT MODE             : COMBINED_WEIGHT
# UVVM:      MEAN CONFIGURED         : false
# UVVM:      MEAN                    : 0.00
# UVVM:      STD_DEV CONFIGURED      : false
# UVVM:      STD_DEV                 : 0.00
# UVVM: ==================================================
```


EmLogic

Brand New **2** - October 2021

- UVVM introduced Specification Coverage in 2020
  - A huge improvement for the whole VHDL Community
  - Allowed the simplest possible way of tracking Requirements
  - A boost for any design where Quality is important
- But - We have also got lots of requests for:
  - More Coverage functionality in UVVM
  - Better integrated verification – than current alternatives in SV and VHDL
  - More understandable expressions and usage
- As a result UVVM is now releasing **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

Functional Coverage – X-B-I

(eXtremely Brief Introduction)

- Functional Coverage
  'is a user-defined metric that measures how much of the design specification has been exercised in verification'
  - Has various functional scenarios been tested.
  - **A manual process is required to set up all wanted scenarios**

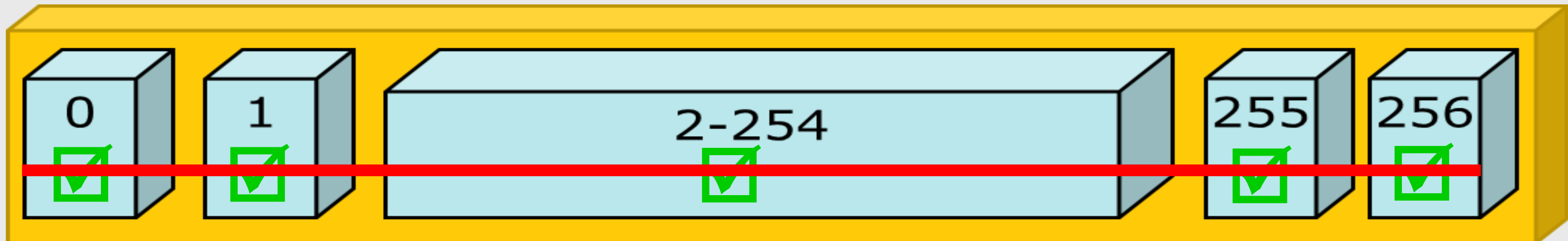
E.g. In a system with a FIFO:
- Has the FIFO been full, - and empty
- Has a write been attempted when full (or read when empty)
- Has the FIFO been full followed by read then write
- etc.

E.g. For a packet oriented protocol (0-256 bytes):
- Has payload size been 0,1,255,256 and something in between
- Has various destination addresses been tested
- Has selected combinations of these been tested

Define Coverpoints (1) (Things you want to check)

- For the given protocol example:
  Make sure that corner case payload sizes have been verified
  - E.g. at least once for each of: 0, 1, 255, 256 and 2-254

A cover point ➔ An actual issue to check



The selected values and ranges are called Bins

  - Then if you generate a packet with payload size of 1 byte,
    → tick off that value
  - Continue making packets of different payload-sizes
    until all values and ranges have been ticked off.

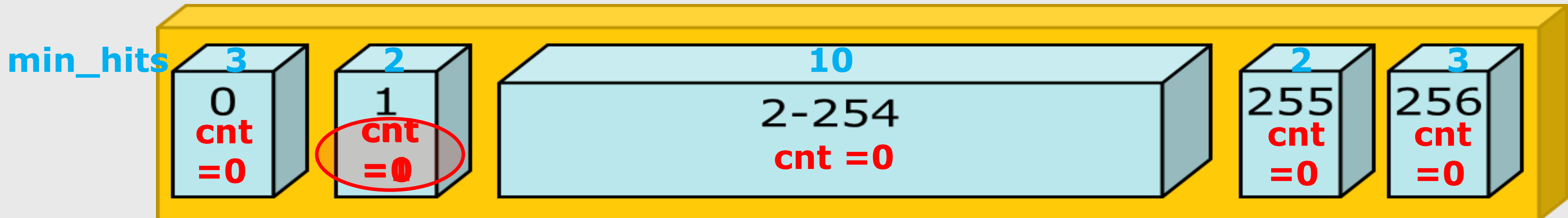
➔ You have now covered all your selected values and ranges

➔ Your Payload-size Coverpoint is covered

Define Coverpoints (2) (Things you want to check)

- For the protocol example:
  Make sure that corner case payload sizes have been verified
  - **E.g. a minimum number of times** for each of: 0, 1, 255, 256 and 2-254

This minimum required number of hits is called min_hits



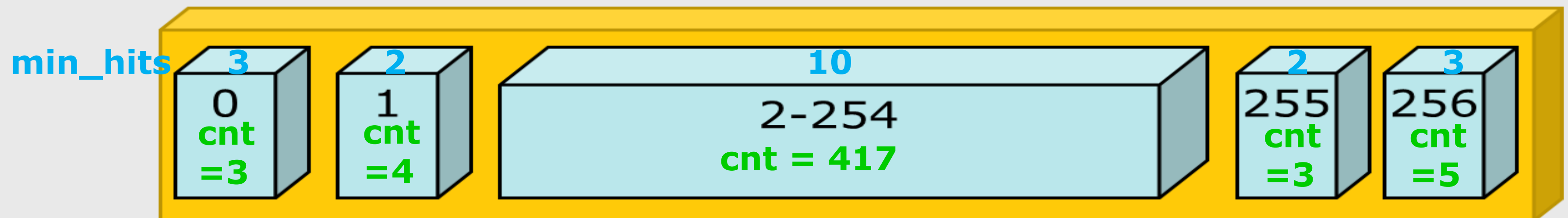
- Then if you generate a packet with payload size of 1 byte,
  → Increase the counter for that bin
- Continue making packets of different payload-sizes until all values and ranges have been applied the minimum number of times required

EmLogic

Define Coverpoints (2) (Things you want to check)

- For the protocol example:
  Make sure that corner case payload sizes have been verified
  - **E.g. a minimum number of times** for each of: 0, 1, 255, 256 and 2-254

This minimum required number of hits is called min_hits



  - Then if you generate a packet with payload size of 1 byte,
    → Increase the counter for that bin
  - Continue making packets of different payload-sizes until all values and ranges have been applied the minimum number of times required .

➔ You have now covered all your selected values and ranges

➔ Your Payload-size Coverpoint is covered

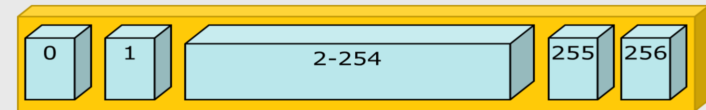
EmLogic

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);
```


EmLogic

Bin generation

```
bin(0)                          -- single bin for single value
bin((2,3,6,8))                  -- single bin for a set of values
bin_range(0,5)                  -- single bin for each value in a range
bin_range(0,5,2)                -- two bins split evenly on range
bin_vector(addr)                -- 2**addr bins
bin_vector(addr,16)             -- 16 bins
```

```
bin_transition((2,4,8))     -- single bin for given sequence
```

```
ignore_bin(7)                -- value to be ignored
ignore_bin_range(5,7)        -- range to be ignored
ignore_bin_transition(4,8) -- sequence to be ignored
```

```
illegal_bin(7)                -- illegal value
illegal_bin_range(5,7)        -- illegal range
illegal_bin_transition(4,7) -- illegal sequence
```

Other Functional coverage features

- `.add_cross()  : Cross coverage for two or more crosses`
- `.is_defined() : To check if bins have been defined`

- Coverage goal modification
  - Specific cover point coverage modification (for bins)
  - or Overall simulation coverage (for all Coverpoint)
  - or both

- Configuration
  - Bin name and scope
  - Alert settings (Illegal bin, Bin overlap)
  - Coverage weight : Weight of CP for overall coverage

- Coverage data base: For accumulation of coverage

EmLogic

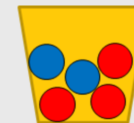
Some reports – out of many

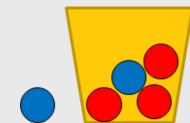
```
# UVVM:  ==========================================================================================
# UVVM:  0 ns *** COVERAGE SUMMARY REPORT (NON VERBOSE): TB seq. ***
# UVVM:  ==========================================================================================
# UVVM:  Coverpoint:             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:  ==========================================================================================
```

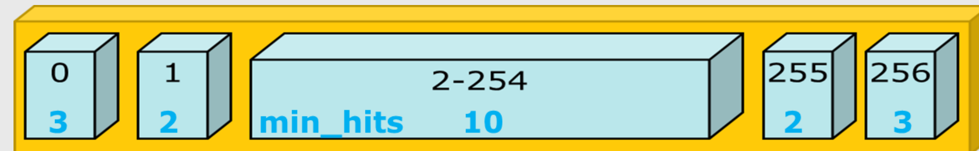
```
# UVVM:  ==========================================================================================
# UVVM:  0 ns *** OVERALL COVERAGE REPORT (VERBOSE): TB seq. ***
# UVVM:  Coverage (for goal 100): Covpts: 50.00%,   Bins: 73.68%,   Hits: 76.00%
# UVVM:  ==========================================================================================
# 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:  ==========================================================================================
```

Optimised randomisation

- Optimised Randomisation is
  - Randomisation without replacement
  - Weighted according to target distribution AND previous events
  - Uses Functional Coverage mechanisms and protected type
    - Target = bins with min_hits







→ the lowest number of randomisations for a given target

→ Major reduction in # packets and thus simulation time

- Is basically Randomisation and Functional Coverage in one
  - Functional need and Use case is Optimised Randomisation
  - Mechanism used is that of Functional Coverage + Weight + Rand

Development and Release of Randomisation and Functional Coverage

- Developed and maintained by Inventas and EmLogic

- Beta version will be released on Github in October
  - As an extension on UVVM utility library
- Active UVVM users may have the Alpha version sooner
  - Send request to et@emlogic.no

EmLogic

Also Brand new: UVVM Steering group

- Founding members are Inventas and EmLogic
  - We have been cooperating on UVVM since January 2021
    - Both on the ESA project and on general UVVM development
- All rights will be given to the UVVM Steering group
  - Copyrights, Github repo, UVVM forum, uvvm.org
- Was founded yesterday…

- Steering group Organisation to be defined
- Steering group to be extended ASAP after that

- We welcome members from the Industy, EDA and Academia
  - **Must** have a strong interest in verification functionality for VHDL
  - **Must** have a good knowledge of and experience with UVVM
  - **Must** want to make UVVM a great tool for the VHDL community

  ➔ Send email to et@emlogic.no and include reason for wanting to join

EmLogic

Design and Verification Courses

- **Advanced VHDL Verification – Made simple**
  - Munich 26-28 October (May change to 5-day online. TBD next week)
- **Accellerating FPGA and Digital ASIC Design**
  - Munich 10-11 November (May change to 3- or 4-day online. TBD soon.)
- 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

https://emlogic.no/courses/

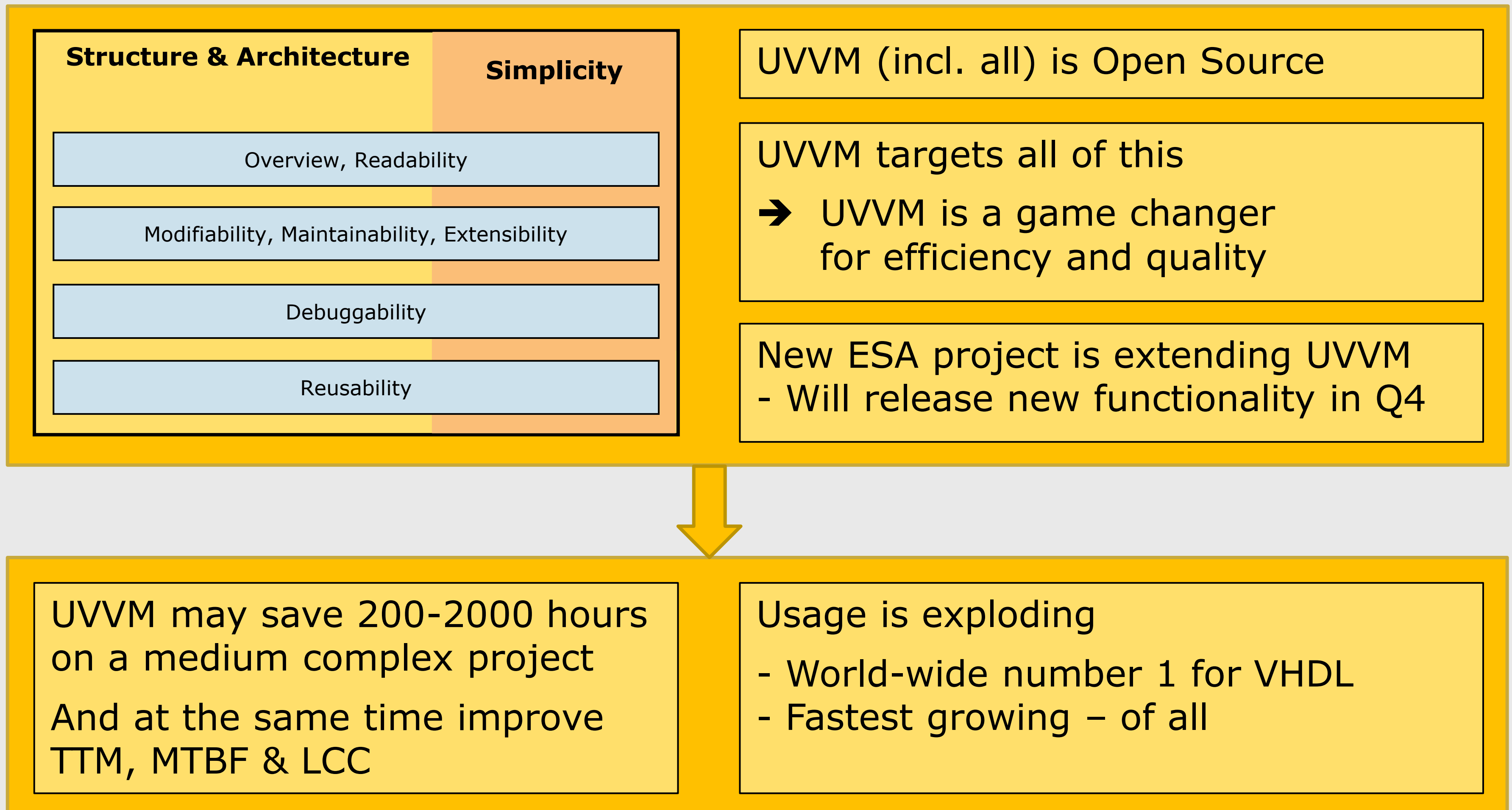
Verification

- 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

- Also includes UVVM CR and FC usage

UVVM in a nutshell

- Huge improvement potential for more structured FPGA verification



EmLogic

EmLogic

Thanks for your attention

Community contributions to UVVM are very welcome…

Please let me know if this would be possible

et@emlogic.no