Coaching UVM Projects

An Individual Approach

Part 1

Dr. Christoph Suehnel

UVM Coach

Agenda: Part 1

- Objectives of this workshop
- What should you tell the customer about UVM?
- Prerequisites
- Objectives of the coaching process
- Planning the coaching project
- How to start an UVM Project
- Setting-up an UVM Project
- Useful UVM Architectures – what you should tell the cust
- Data structure for UVM Projects

Objectives Of This Workshop

- To enable individuals to coach and drive UVM projects effectively
- To give customers the best of UVM
- How to train newbees in UVM
- To enable consultants to deploy UVM in a streamlined way

What should you tell the customer about UVM?

- What is UVM?
  - Highly complicated
  - Hard to understand
  - Ineffective for small designs
  - Not useful for FPGA customers

- What is UVM?
  - A well organized methodology for verification
  - A construction set for generating testbenches
  - Easy to handle
  - Useful for any kind and complexity of design
  - A methodology to speed-up testbench generation
  - Supporting reuse highly effective
  - Making verification controllable
  - Improving verification quality
  - Helps meeting time-to-market

Prerequisites

- Knowledge about verification
- HDL know-how
- SystemVerilog
- Transaction Level Modelling
- UVM Knowledge
- Simulation know-how

Objectives of the coaching process

- To enable customers to employ UVM
- To train customers on UVM
- To help implementing UVM-based testbenches
- To give the right understanding of separating verification environments from tests
- How to resolve issues in the UVM implementation/execution process

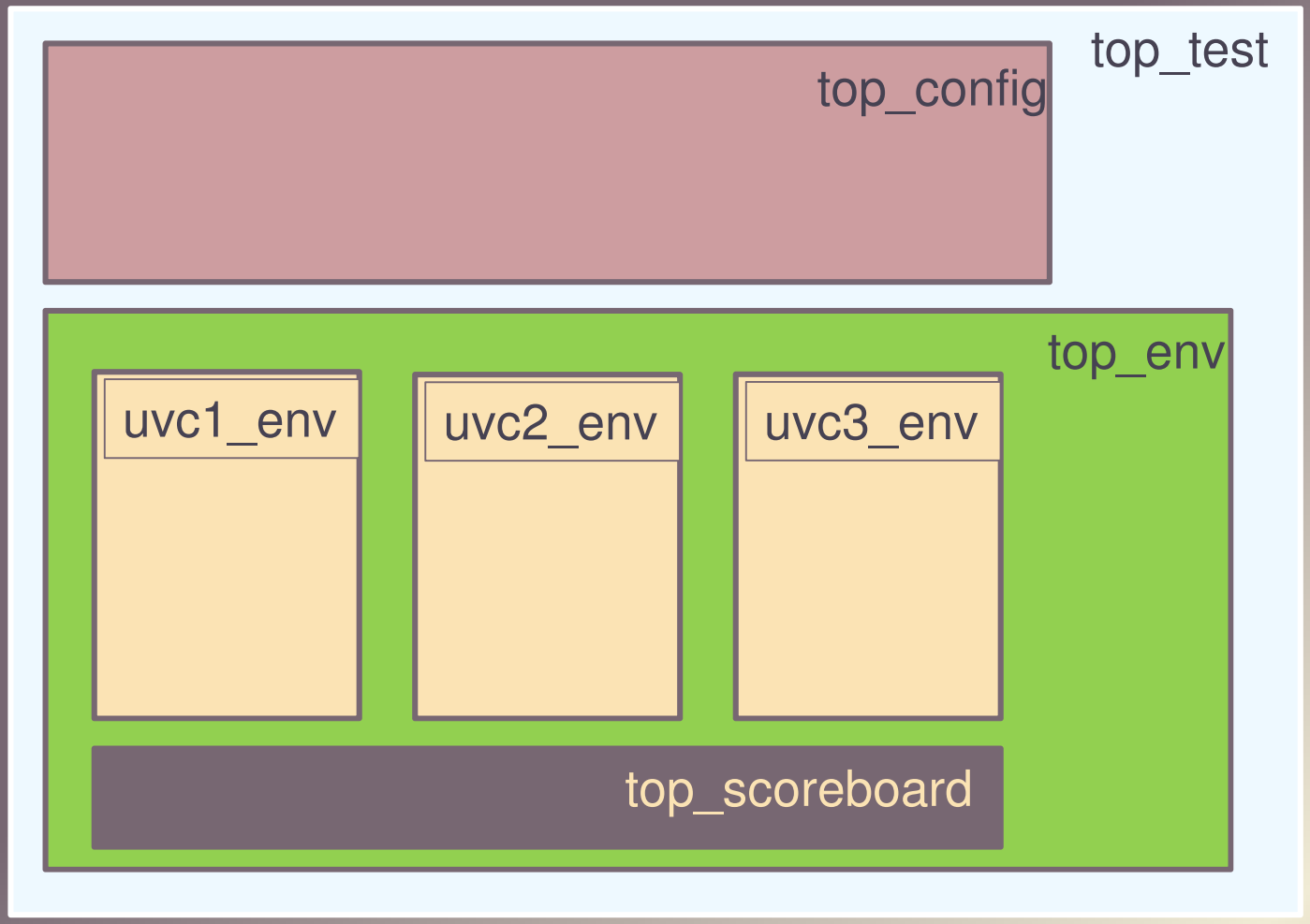
Planning the coaching project

- Asking the customer the key questions
  - What are the functional interfaces of the design?
  - How many there are?
  - Does the design have registers?
  - How many?
  - What is the main functionality?
    - Which data have to be compared?
  - What about a reference model?
    - Is it needed?
    - Does it exist?
    - How is it implemented (language)?
- Making your decisions
  - Fine-tuning of architecture
  - Planning
  - Scheduling

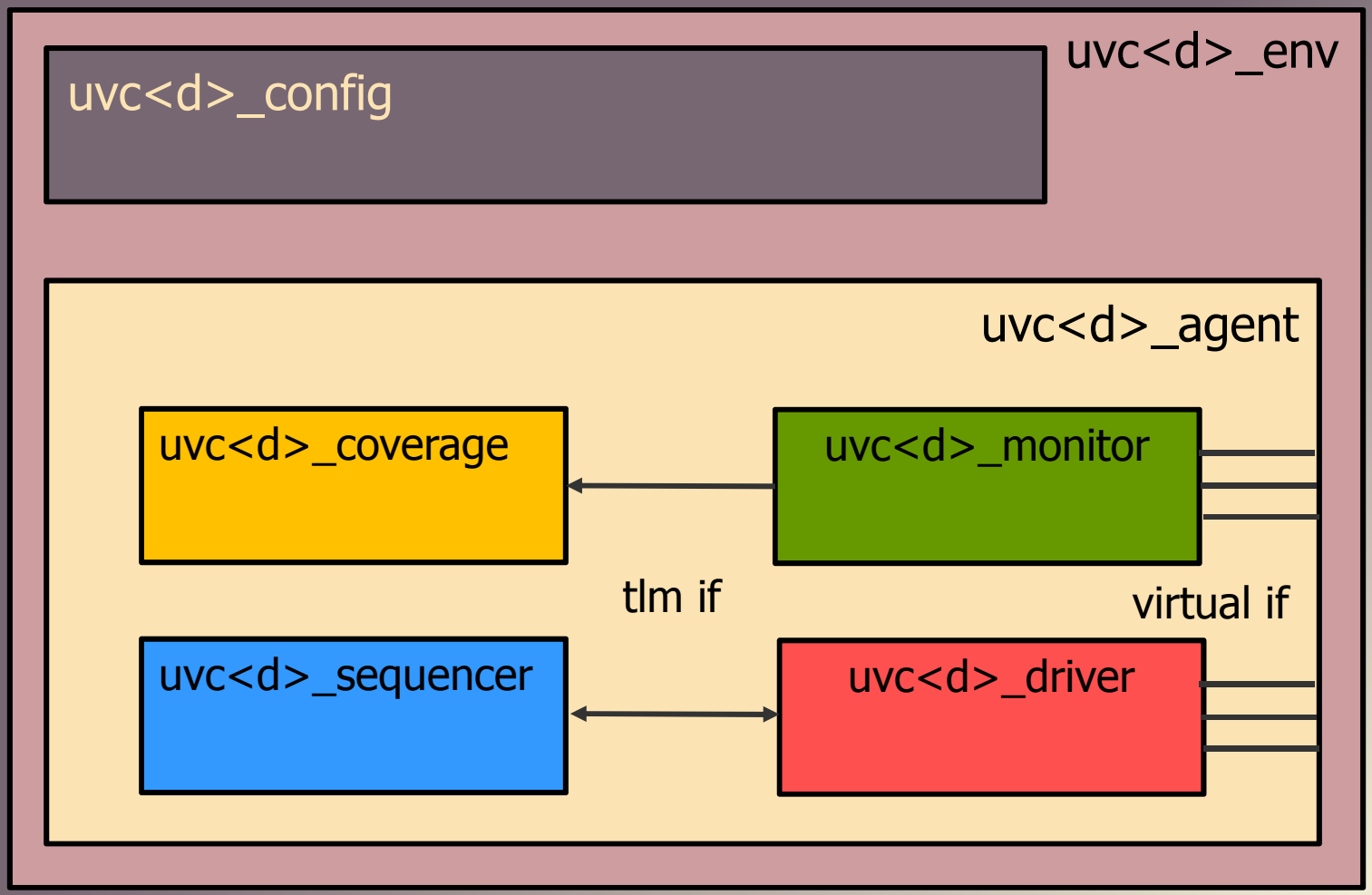
Useful UVM Base Architectures

- Use a well-structured approach for the UVM testbench
- Agent or environment based for each functional interface
- Personally I prefer an environment-based UVM testbench approach
- The env contains the agent
- The top environment contains the env ‘s of the functional interfaces

Useful UVM Base Architectures - top_test



Useful UVM Base Architectures - uvc<d>_env



How to start an UVM Project

- Important is to generate a quick success
- Motivates the customer
- Reviewing all data you got so far from the customer
- Define a start date together with the customer
- Setup a kickoff meeting, invite the customer's management
- Provide the schedule to the customer
- Check the customer working environment (licenses, working platform etc.)
- Define the first work packet and communicate it to the customer

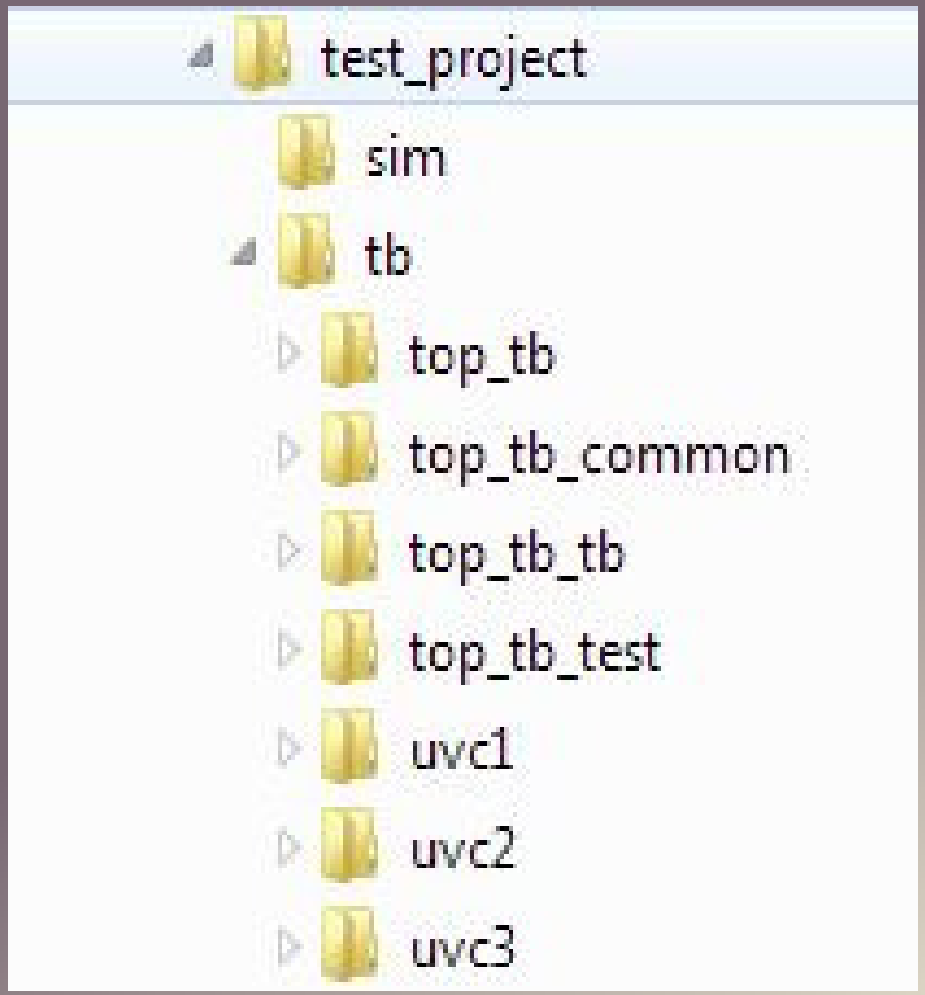
Setting-up a UVM Project

- How would you start?
- It is boring to start with textual descriptions/specifications only
- Gives the customer a really bad feeling
- Start with a big jump, giving the customer the right feeling
- Provide a complete base environment at the kickoff meeting
  - Fitting to the DUT
  - Compilable and runnable from the beginning
  - Saves a huge amount of time
  - Prevents from doing a lot of basic SV syntax errors

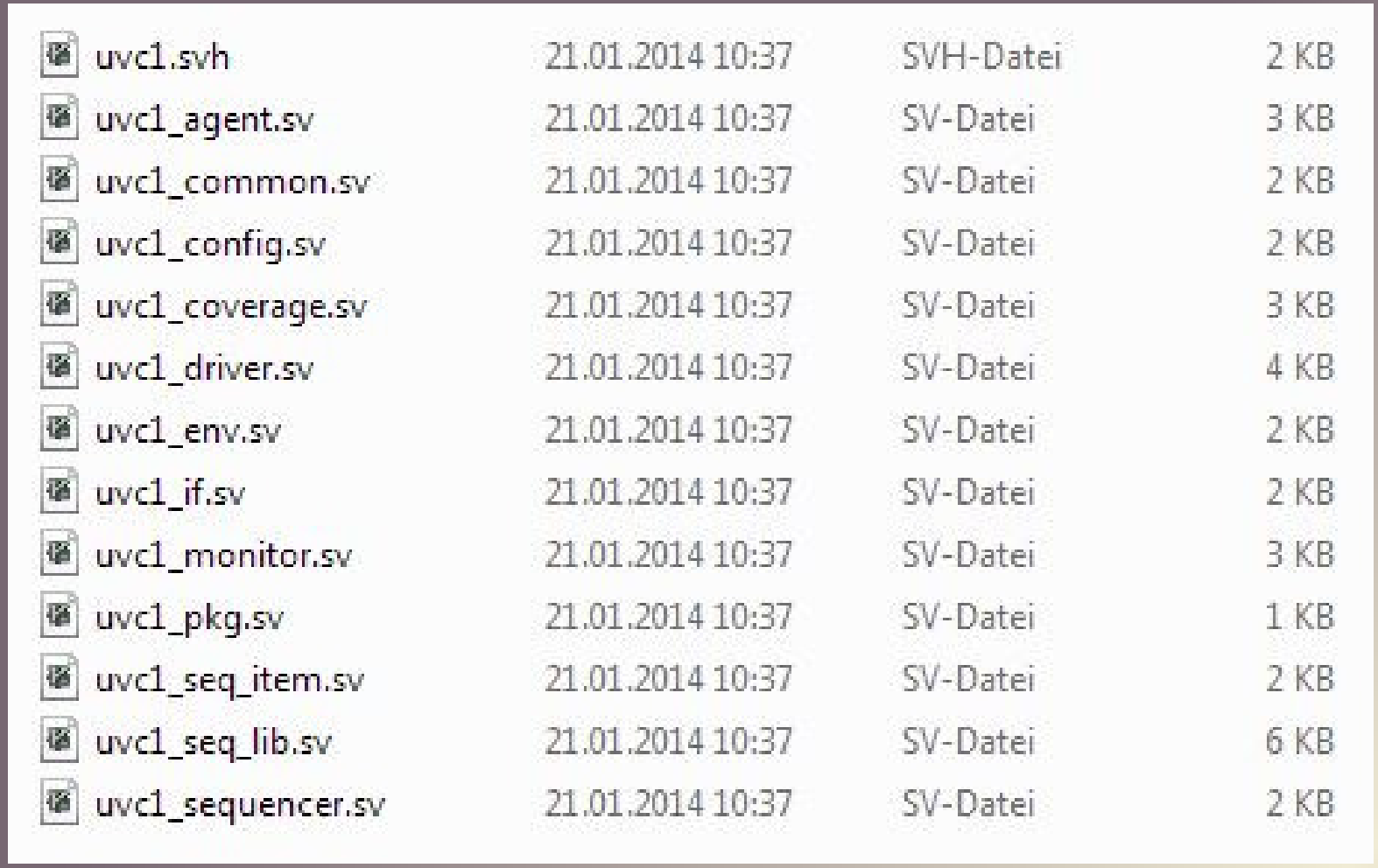
Setting-up an UVM Project

- Automatic generation of a customized UVM environment
- UVM Framework generator: **uvm_skel_gen.pl**
- Based on the juvb Perl script distributed in the UVM-World contributions
- Extended from a simple UVC generator to an UVM testbench generator

Possible Data Structure for UVM Projects



Possible Data structure for UVM Projects



uvc1.svh	21.01.2014 10:37	SVH-Datei	2 KB
uvc1_agent.sv	21.01.2014 10:37	SV-Datei	3 KB
uvc1_common.sv	21.01.2014 10:37	SV-Datei	2 KB
uvc1_config.sv	21.01.2014 10:37	SV-Datei	2 KB
uvc1_coverage.sv	21.01.2014 10:37	SV-Datei	3 KB
uvc1_driver.sv	21.01.2014 10:37	SV-Datei	4 KB
uvc1_env.sv	21.01.2014 10:37	SV-Datei	2 KB
uvc1_if.sv	21.01.2014 10:37	SV-Datei	2 KB
uvc1_monitor.sv	21.01.2014 10:37	SV-Datei	3 KB
uvc1_pkg.sv	21.01.2014 10:37	SV-Datei	1 KB
uvc1_seq_item.sv	21.01.2014 10:37	SV-Datei	2 KB
uvc1_seq_lib.sv	21.01.2014 10:37	SV-Datei	6 KB
uvc1_sequencer.sv	21.01.2014 10:37	SV-Datei	2 KB

Data structure for UVM Projects - uvc<d>_pkg

```
 1 //=============================================================================
 2 // Copyright(c) Dr. Christoph Suehnel 2014 confidential, all rights reserved
 3 //=============================================================================
 4 // Project  : test_project
 5 //
 6 // File Name: uvc1_pkg.sv
 7 //
 8 // Author   : Name   : Christoph Suehnel
 9 //            Email  : christoph@christoph-suehnel.de
10 //            Tel    : +49 8822 835 9239
11 //            Company: Dr. Christoph Suehnel
12 //            Dept   : Consulting
13
14 // Version:
15
16 //=============================================================================
17 // Description:
18 //
19 // Overall package for UVC uvc1
20 //
21 //=============================================================================
22
23   package uvc1_pkg;
24
25     `include "uvc1.svh"
26
27   endpackage : uvc1_pkg
28
29   `include "uvc1_if.sv"
30
```

Data structure for UVM Projects - uvc<d> includes

```
22
23 `ifndef UVC1_SV
24 `define UVC1_SV
25
26   // Imports
27   import uvm_pkg::*;
28
29   // General includes
30   `include "uvm_macros.svh"
31
32   // UVC includes
33   `include "uvc1_common.sv"
34   `include "uvc1_seq_item.sv"
35   `include "uvc1_config.sv"
36   `include "uvc1_driver.sv"
37   `include "uvc1_monitor.sv"
38   `include "uvc1_sequencer.sv"
39   `include "uvc1_coverage.sv"
40   `include "uvc1_agent.sv"
41   `include "uvc1_env.sv"
42   `include "uvc1_seq_lib.sv"
43
44 `endif // UVC1_SV
45
```

Data structure for UVM Projects - uvc<d>_agent

```
26 //------------------------------------------------------------------------------
27 class uvc1_agent extends uvm_agent;
28 //------------------------------------------------------------------------------
29
30   uvm_active_passive_enum is_active = UVM_ACTIVE;
31   uvc1_sequencer sequencer;
32   uvc1_driver driver;
33   uvc1_monitor monitor;
34   uvc1_coverage coverage;
35
36   `uvm_component_utils_begin(uvc1_agent)
37     `uvm_field_enum(uvm_active_passive_enum, is_active, UVM_ALL_ON)
38   `uvm_component_utils_end
39
40   extern function new(string name = "uvc1_agent", uvm_component parent);
41   extern function void build_phase(uvm_phase phase);
42   extern function void connect_phase(uvm_phase phase);
43
44 endclass : uvc1_agent
45
46 function uvc1_agent::new(string name = "uvc1_agent", uvm_component parent);
47    super.new(name,parent);
48 endfunction : new
49
50 function void uvc1_agent::build_phase(uvm_phase phase);
51   super.build_phase(phase);
52   `uvm_info(get_type_name(), $psprintf("agent is: %s", is_active), UVM_MEDIUM)
53   monitor = uvc1_monitor::type_id::create("monitor",this);
54   coverage= uvc1_coverage::type_id::create("coverage",this);
55   if (is_active == UVM_ACTIVE)
56     begin
57       driver    = uvc1_driver::type_id::create("driver",this);
58       sequencer = uvc1_sequencer::type_id::create("sequencer",this);
59     end
60 endfunction : build_phase
61
```