

An Easy VE/DUV Integration Approach

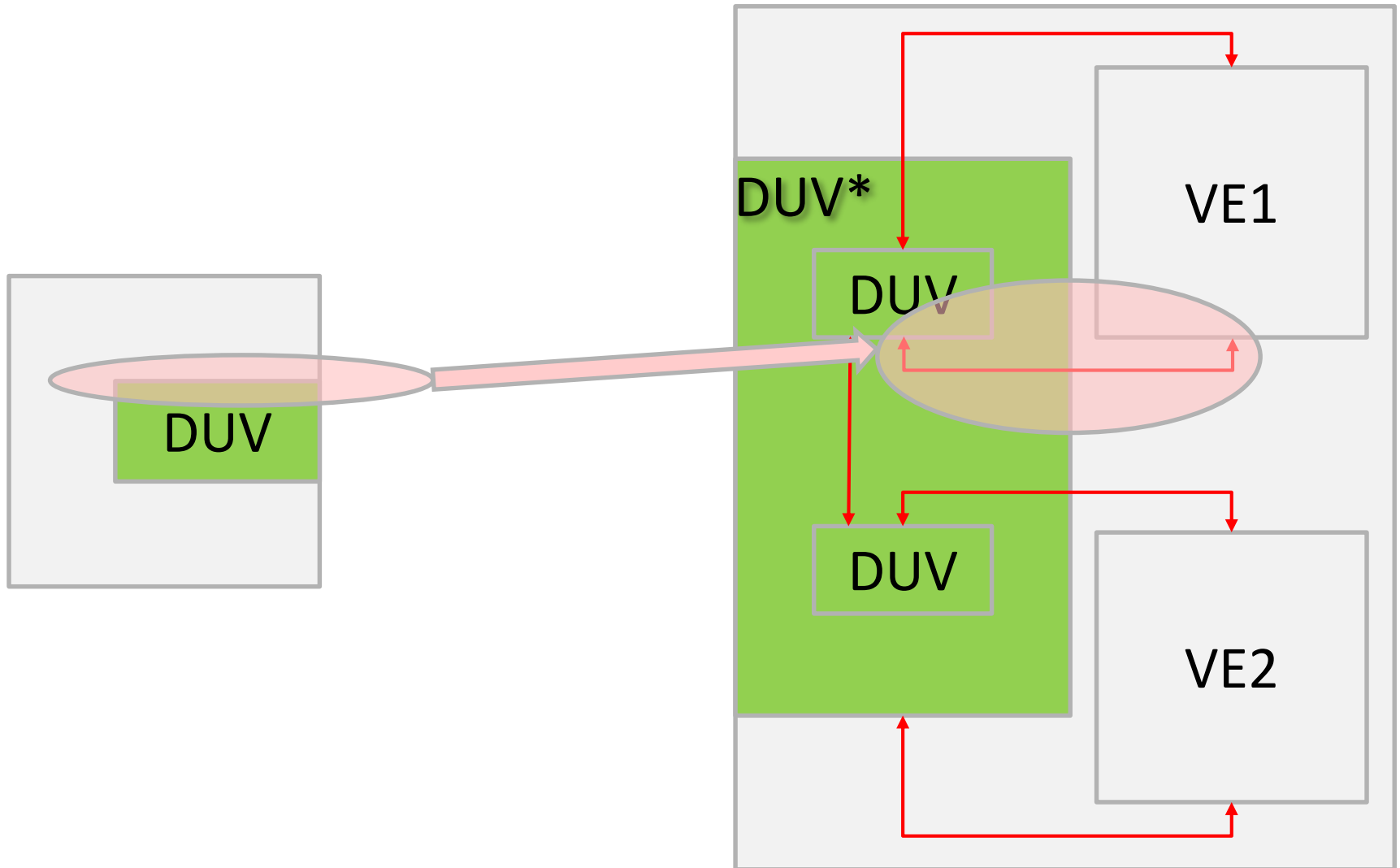
Uwe Simm – Cadence Design Systems, Inc.

The Cadence logo features the word "cadence" in a lowercase, bold, sans-serif font. A small red horizontal bar is positioned above the letter "a". A registered trademark symbol (®) is located at the top right of the letter "e".

Is There a Problem?



TB Vertical Reuse



Common Methods

- Methods to distribute interface from TB into VE
 - Custom function distributing interfaces such as **assign_vi()**
 - Config struct member
- Methods to access interface in DUV
 - “wire” (TB) interface to primary DUV port
 - Create interface in TB and assign elements by OOMR
 - If interface already exists access via OOMR

Common Issues

- DUV code modifications for verification
- Simple adjustments might lead to extensive source changes (OOMRs are not reusable)
- Knowledge and code in various places
 - Which interfaces exist?
 - Which component accesses which interface?
 - How is the interface retrieved and checked?

Can we do better?

What if?

- Not adding verification infrastructure to DUV
 - Not “wiring” interface through DUV ports to top
 - Make interface instance(s) in the DUV on demand without touching the code
- Make all interfaces available in a central place via a key
- Query interfaces from test bench as needed

3 Piece Solution

- SystemVerilog **bind** construct
- Interface self registration
- Central database

SystemVerilog “bind”

- **bind** can make an instance in a module
 - In all instances of a type
 - Or in a particular instance only
- No code change required in target module

```
// -*- binds an instance of the "clk_intf" into "sub_block"
with instance
// name "clk_intf_i" signal names are bound by name
bind sub_block clk_intf clk_intf_i(.*);

// by instance
bind top_block.sub_block_i2 clk_intf clk_intf_i(.*);
```


Interface Self Registration

- Interface instance actively registers itself
 - In central db
 - Every interface instance can do it automatically (no code required outside of interface)
 - Key = verilog instance path of interface instance
 - Value = reference to current interface
- Registration is automatic during startup

Interface Self Registration

```
interface clk_intf(output bit clk);  
    // interface wires,regs,tasks,functions ...  
    // -*- mandatory code for self registration  
function automatic void register();  
    virtual clk_intf vif;  
`ifdef INCA // wrt mantis4300  
    vif = clk_intf;  
`endif  
    cdns_vif_db#(virtual clk_intf)::register_vif(  
                                                vif, $sformatf("%m"));  
endfunction  
  
    initial register();  
    // -*- mandatory code ends  
endinterface
```

Self Reference to Interface

- Like `this` for class instances
- Not covered by current LRM
- <http://www.eda.org/mantis/view.php?id=4300>
- Major vendors support it
 - unfortunately with a different syntax
- Only a single line difference
 - contained change

Interface Registry

- Any typed key-value store will do
- Package uses `uvm_config_db` as database
 - Provides overrides, trace, dump, storage, types
 - UVM users know `uvm_config_db`
- Database can be extended to offer addon services (debug, reporting)

```
class cdns_vif_db#(type T=int) extends uvm_config_db#(T);  
  static function void register_vif(T vif, string vifName);  
  static function void retrieve_vif(ref T vif, input  
    uvm_component cntxt, string path, bit validate=1);  
endclass
```

Approach provided so far

- Interface instances can be made
 - Without code change in DUV
 - At any DUV level
- All interface instances available in DB
- Database can provide add-on services
 - Statistics
 - Logging
 - Common checking and retrieval code
 - Debugging aid

Verification Env Use Model

- Rule: #1 make IF instance; #2 retrieve IF
- Retrieve interface instance simply via the key

```
class testbench extends uvm_env;
    // -*- this is a container private virtual interface
    virtual clk_intf vif;

    function void build();
        super.build();
        cdns_vif_db#(virtual
            clk_intf)::retrieve_vif(vif,this,"clk_intf_i");
    endfunction
```

Module-to-System Use Model

- VE topology usually matches DUV topology
 - so every TB component has an associated DUV instance
 - Package assumes that for every VE component an **“HDLContext”** can be constructed
- Full **“HDLContext”** for a TB component matches Verilog instance path of associated DUV hierarchy
 - Path fragments stored as property in `uvm_config_db`
- Key for lookup is
 - **HDLContext** for context component
 - Plus name during `request_vif()`

Example

```
// assumption: tb1,tb2 are children of top
uvm_config_db#(string)::set(top,"","HDLContext","top");
uvm_config_db#(string)::set(tb2,"","HDLContext","sub_block_i2");

uvm_config_db#(string)::set(tb1,"","HDLContext","sub_block_i1");

// the query for clk_intf_i via
cdns_vif_db#(virtual
clk_intf)::retrieve_vif(vif,this,"clk_intf_i");

// would return the interface for this=tb2
"top.sub_block_i2.clk_int_f"

// and for this=tb1
"top.sub_block_i1.clk_int_f"
```


Summary

- Presented an easy path to integrate DUV/TB
- Path provides
 - no DUV changes or special structure for verification required
 - All IF instances available in central place
 - Each TB component can query IF
 - Support for horizontal/vertical reuse
- Code for “interface registry package” can be downloaded from <http://forums.accellera.org/files/>

Thank you

Q&A