Verification Futures

The Verification Future needs an Easier™UVM

John Aynsley, Dr Christoph Suehnel, Francois Cerisier

The Verification Future needs an Easier™UVM

- Motivation
- Introducing *Easier UVM*
- Coding Guidelines
- Code Generation

DOULOS

easier UVM

SystemVerilog for Constrained Random

UVM

UVM = Universal Verification Methodology

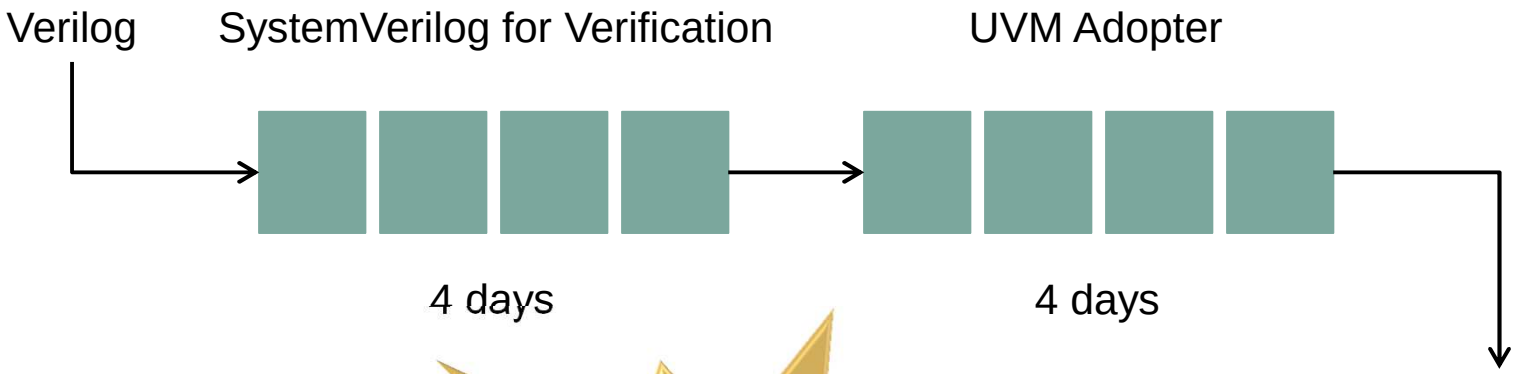
- UVM is actively supported by all major vendors

- Drives mutually consistent SystemVerilog implementations
- One standard enhances interoperability and ecosystem growth
- Increases confidence for SystemVerilog adoption

easier UVM

UVM Itself is Challenging

- Doulos training



Significant up-skilling

What next?

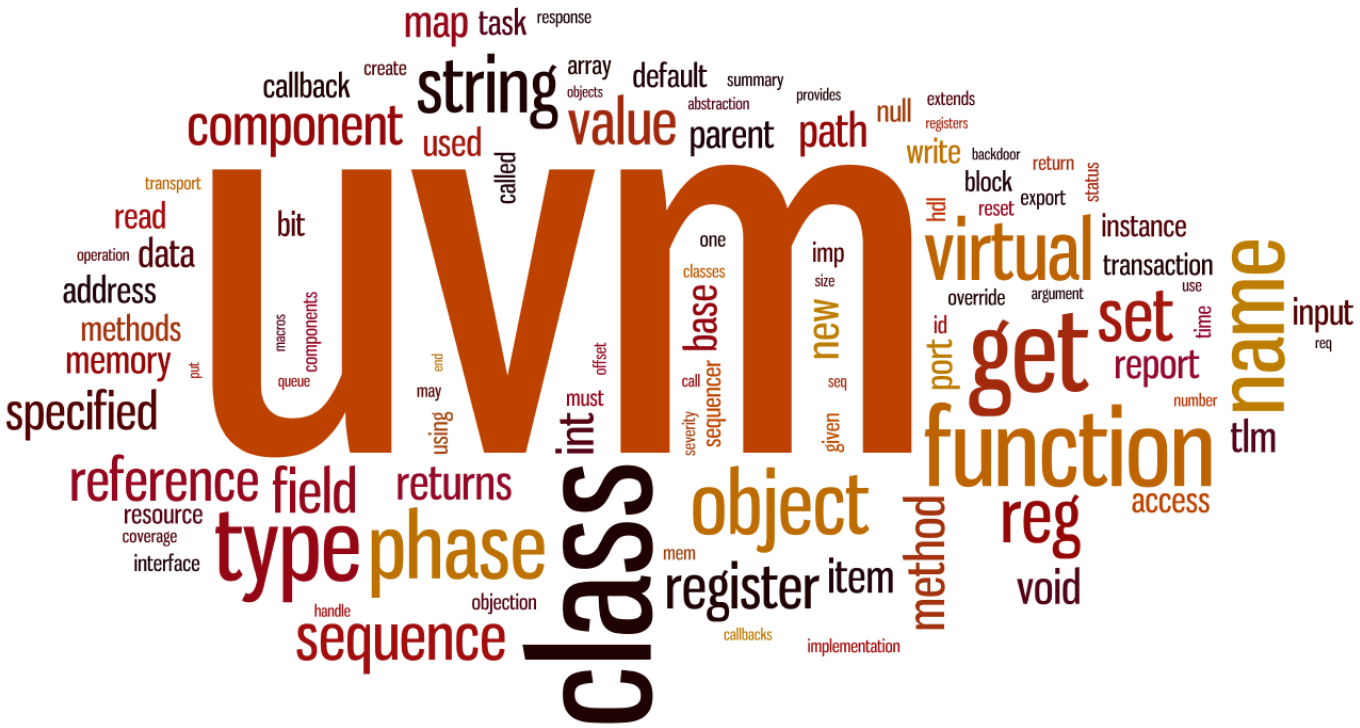
Tools?

Coaching?

Guidelines?

easier UVM

UVM Itself is Challenging



- ~ 300 classes in UVM Base Class Library
- The UVM documentation does not answer all the questions
- There are many ways to do the same thing - need a methodology

easier UVM

The Verification Future needs an Easier™UVM

- Motivation
- Introducing *Easier UVM*
- Coding Guidelines
- Code Generation

DOULOS

easier UVM

Easier UVM – First Version (2011)

- Aimed at mainstream Verilog & VHDL users
- Goal = Reduce UVM to a set of simple concepts and coding idioms
- *Easier UVM* is UVM
- Use more features of UVM as you learn
- Learning UVM is still not easy...

easier UVM

Easier UVM – Second Version

- Same principles, plus ...
- Specific coding guidelines – "One way to do it"
- Automatic code generator

Benefits

- Help individuals and project teams
  - learn UVM and avoid pitfalls
  - become productive with UVM (saves ~ 6 weeks)
  - use UVM consistently
- Reduce the burden of supporting UVM code

Easier UVM Code Generator

- Written in Perl
- Will be released with an open source license
- Generates code conforming to the Easier UVM guidelines
- Can be modified

easier UVM

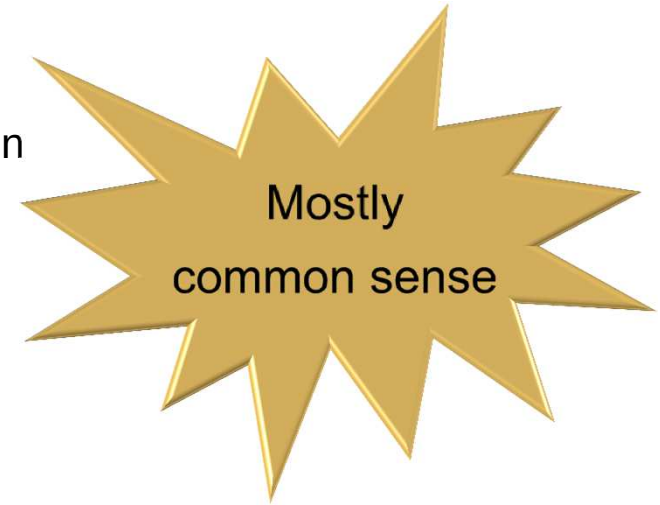
The Verification Future needs an Easier™UVM

- Motivation
- Introducing *Easier UVM*
- Coding Guidelines
- Code Generation

easier UVM

Coding Guidelines

- Lexical Guidelines and Naming Conventions
- General Guidelines
- General Code Structure
- Clocks, timing and synchronization
- Transactions
- Sequences
- Objections and Sequences
- Components
- Connection to the DUT
- TLM Connections
- Configurations
- The Factory
- Tests
- Messaging
- Functional Coverage
- The Register Layer
- Agent Data Structure and Packaging



Coding Patterns

Pattern 1

```
class my_comp extends uvm_component;
  `uvm_component_utils(my_comp)

  function new(string name, uvm_component parent);
    super.new(name, parent);
  endfunction

  function void build_phase(...);
    ...
endclass
```

- The order of the declarations
- Specific naming conventions
- Which macros to use
- Which methods to override
- Coding patterns for specific situations

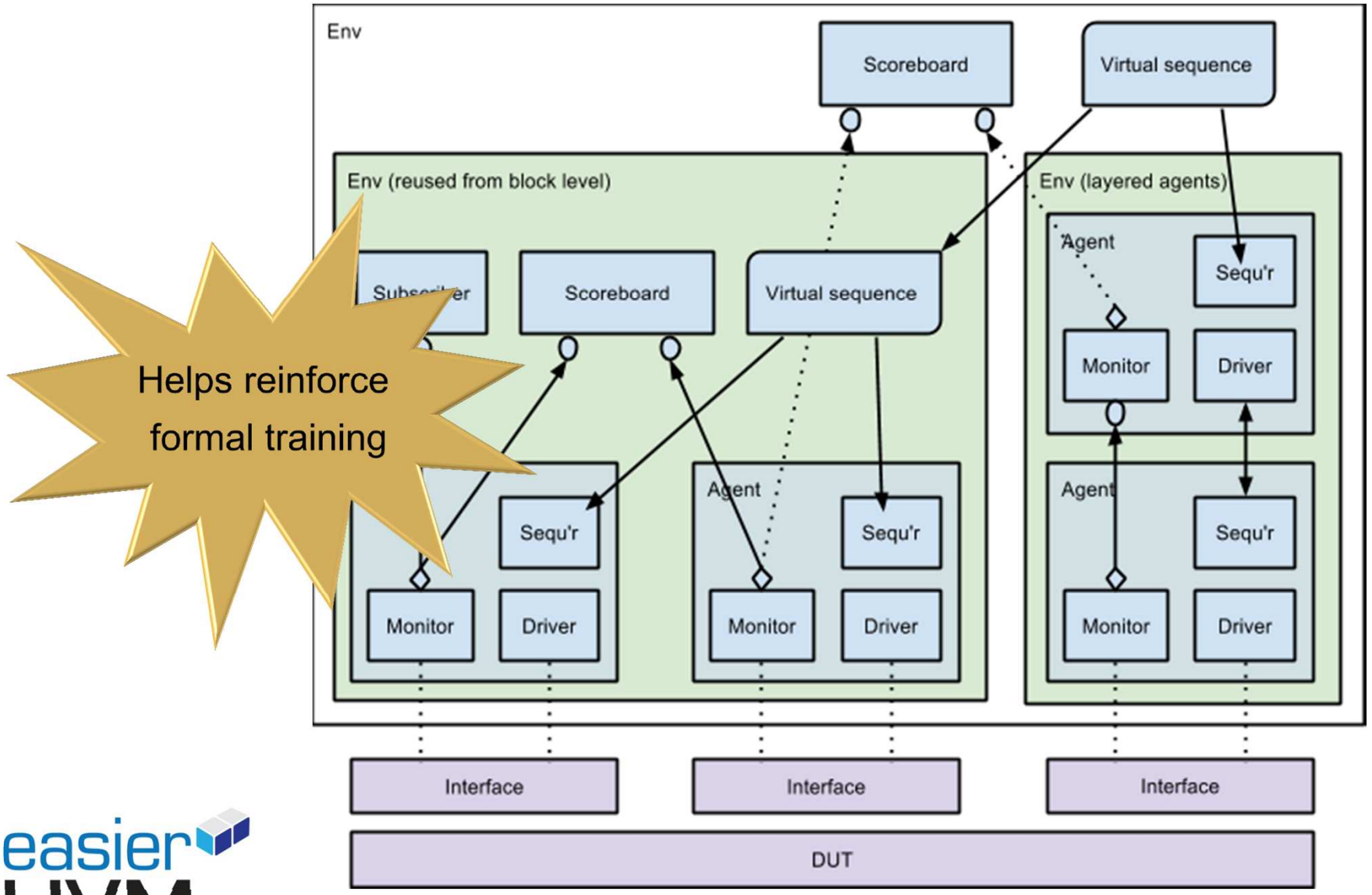
Pattern 2a

```
class my_tx extends uvm_sequence_item;
  `uvm_object_utils(my_tx)

  function new (string name = "");
    super.new(name);
  endfunction

  function string convert2string;
    ...
endclass
```

Pattern 2b

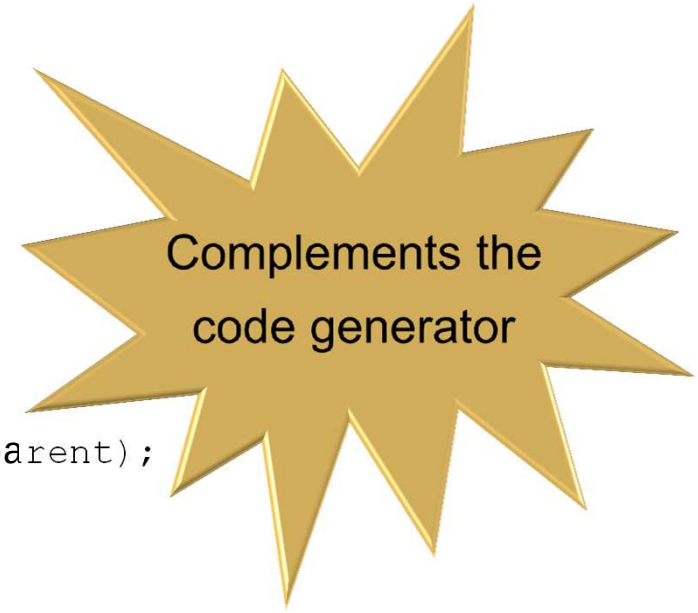
```
class my_seq extends uvm_sequence #(my_tx);
  `uvm_object_utils(my_seq)

  function new(string name = "");
    super.new(name);
  endfunction
  ...
  task body;
    ...
endclass
```

How to Structure a Verification Env



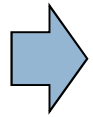
Examples

```
class my_component extends uvm_env;
 `uvm_component_utils(my_component)

 // Transaction-level ports and exports
 uvm_analysis_port #(my_tx) a_port;

 // Virtual interfaces
 virtual dut_if vif;

 // Internal data members (variables)
 my_agent m_agent;

 // Constructor
 function new (string name, uvm_component parent);
   super.new(name, parent);
 endfunction

 // Standard phase methods
 function void build_phase(uvm_phase phase);
   a_port  = new("a_port", this);
   m_agent = my_agent::type_id::create("m_agent", this);
 endfunction
 ...
```



The Verification Future needs an Easier™UVM

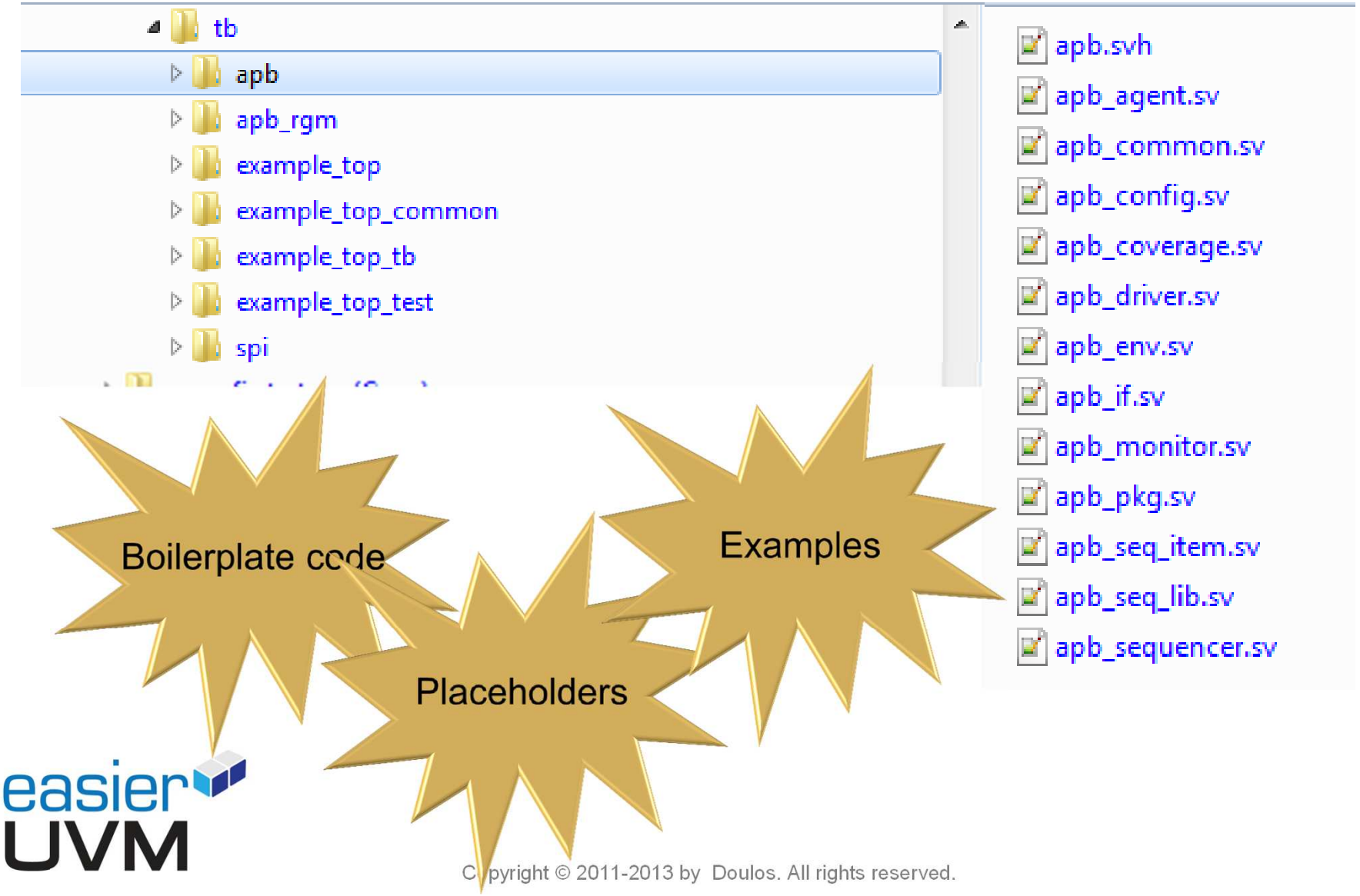
- Motivation
- Introducing *Easier UVM*
- Coding Guidelines
- Code Generation



DOULOS

easier UVM

Output from Generator



Boilerplate Code

```
`ifndef SPI_SEQ_ITEM_SV
`define SPI_SEQ_ITEM_SV

class spi_seq_item  extends uvm_sequence_item;

`uvm_object_utils(spi_seq_item)

// class properties
rand logic [127:0] data;
rand bit [6:0] no_bits;
rand bit RX_NEG;

extern function new(string name="spi_seq_item");
extern function void do_copy(uvm_object rhs);
extern function bit  do_compare(uvm_object rhs, uvm_comparer comparer);
extern function string convert2string();
extern function void do_print(uvm_printer printer);
extern function void do_record(uvm_recorder recorder);

endclass : spi_seq_item


function spi_seq_item::new(string name = "spi_seq_item");
  super.new(name);
endfunction : new

........
```

Placeholders

```
task spi_driver::run_phase(uvm_phase phase);
  // add additional declarations here
  super.run_phase(phase);
  `uvm_info(get_type_name(),"run_phase",UVM_MEDIUM)
  // set signals on reset values here
  @(posedge vif.reset) // reset goes inactive
  forever begin
    seq_item_port.get_next_item(req);
    @(posedge vif.clk)
    `uvm_info(get_type_name(), {"req item\n",req.sprint}, UVM_MEDIUM)
    // insert the driver protocol here
    $cast(rsp, req.clone());
    // adopt the rsp
    seq_item_port.item_done();
  end
endtask : run_phase
```

easier UVM

Examples


```
module example_top_tb;
  ...
  apb_if    apb_if_i();
  spi_if    spi_if_i();

  initial
    begin
      uvm_config_db #(int)::set(null, "*", "recording_detail", 1);
      uvm_config_db #(virtual apb_if)::set(null, "uvm_test_top", "apb_if_i", apb_if_i);
      uvm_config_db #(virtual spi_if)::set(null, "uvm_test_top", "spi_if_i", spi_if_i);

      run_test();
    end

  always #10 clock = ~clock;

  initial
    begin
      clock=0;
      reset=1;          //active high reset for this example
      #75 reset=0;
    end
endmodule
```

Delivering Know-How www.doulos.com

Hardware Design

» VHDL » Verilog » SystemVerilog
» Altera » Microsemi » Xilinx

Embedded Systems and ARM

» C » C++ » UML » RTOS » Linux
» ARM Cortex A/R/M series

ESL & Verification

» SystemC » TLM-2.0 » SystemVerilog
» OVM/VMM/UVM » Perl » Tcl/Tk