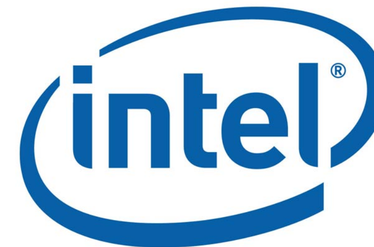
UVM-SystemC: Migrating complex verification environments

Stephan Gerth, Fraunhofer IIS/EAS

Akhila Madhukumar, Intel India Pvt. Ltd

Fraunhofer

IIS



Agenda

- Preface: UVM-SystemC standardization update
- Main: Migrating complex verification environments
  - Previous environment
  - Randomziation using SCV & CRAVE
  - Results
- Q&A

PART 0: UVM-SYSTEMC STANDARDIZATION UPDATE

Updates since DVCon Europe 2016

- Register API: Basic backdoor implemented (no DPI)
- SystemC 2.3.2 compatibility
  - Header includes
  - Pkg-config
  - Immediate notification mechanism
- UVM 1.2 Reporting API
- Stability review

© Accellera Systems Initiative

Current status

- Effort currently shared within small group
  - More input from interested parties welcome and needed
  - Man power needs to be increased for faster development
- Preview release
  - Final packaging & testing
  - shortly after DVCon Europe 2017

© Accellera Systems Initiative

Plans for 2018

- Improve API compatibility to IEEE 1800.2-2017
- Complete Register API (frontdoor/backdoor)
- Simplify CRAVE integration
- Smart Pointer implementation
  - Main branch: make API more clear about ownerships
  - Separate branch: implement shared pointers in API
- Add more examples
  - Ubus
  - Codec
- User Guide

PART 1: PREVIOUSLY USED VERIFICATION ENVIRONMENT

© Accellera Systems Initiative

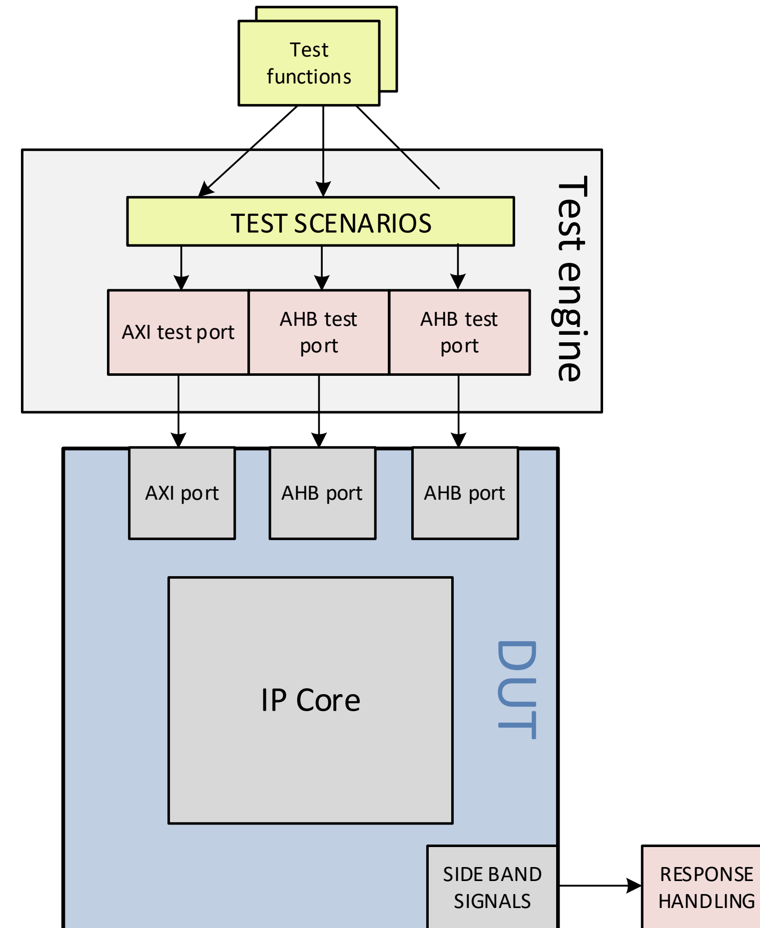
Overview

- Previous verification environment
- Limitations and issues
- Migration details
  - Layers
  - Driver & Monitors
  - Transactions

© Accellera Systems Initiative

Conventional SystemC Testbench

- Custom test functions & test engine
  - test functions (test ports) created to mimic bus transaction drivers; one per transaction type
  - Individual functions for each test scenarios
- Distinct methods for writes and reads
  - Number of methods depends on types of valid write/read as per the protocol being implemented viz. single, burst, posted or non-posted
- Self-checking and parametrized tests
- Additional task created for running multiple tests in parallel from different interfaces
  - Vector classes used to keep track of test functions being launched from each interface



© Accellera Systems Initiative

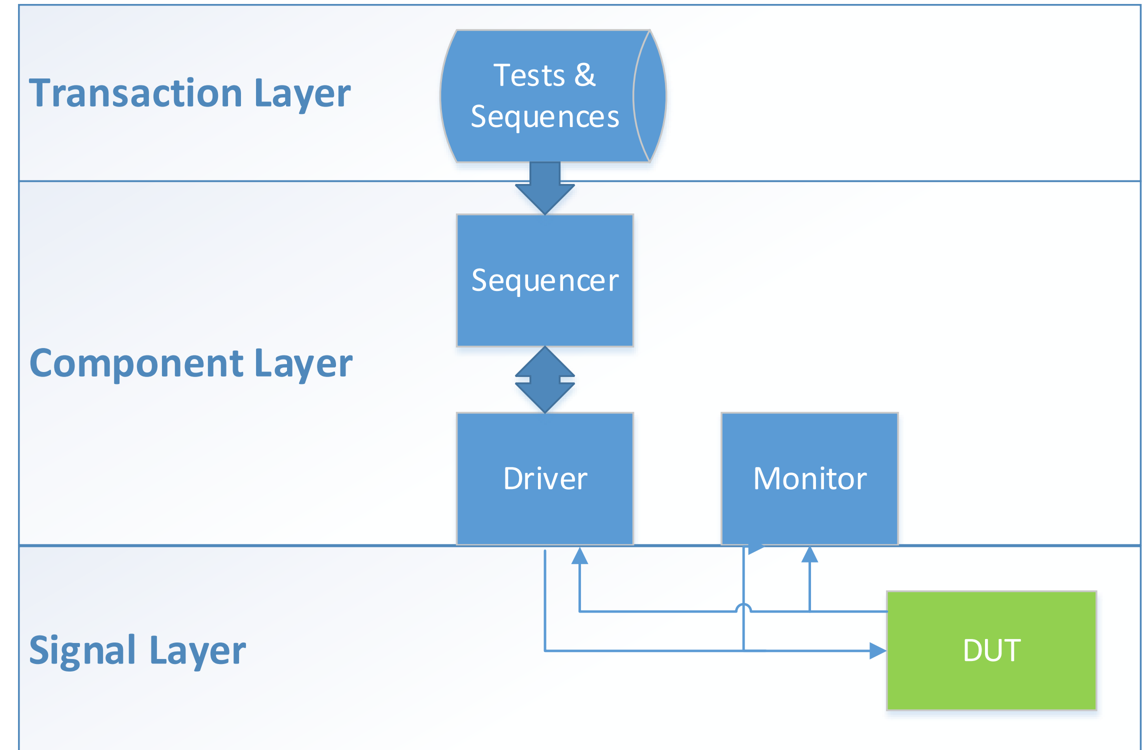
Limitations and potential reasons for UVM-SystemC

- Limitations of Non-UVM-SC testbench
  - Higher learning curve for new user as TB has no standardized architecture
  - Minimal reuse of tests/components across projects
  - Configurability of testbench is limited
  - Inadequate constrained randomization
  - Narrow scope of IP to SoC reuse (SoC usually has UVM-SV based framework)
- Reasons for UVM-SC adoption:
  - Re-usability
  - Configurability
  - Constrained randomization
  - Standardization across languages
  - Easier adoption for UVM-SV users

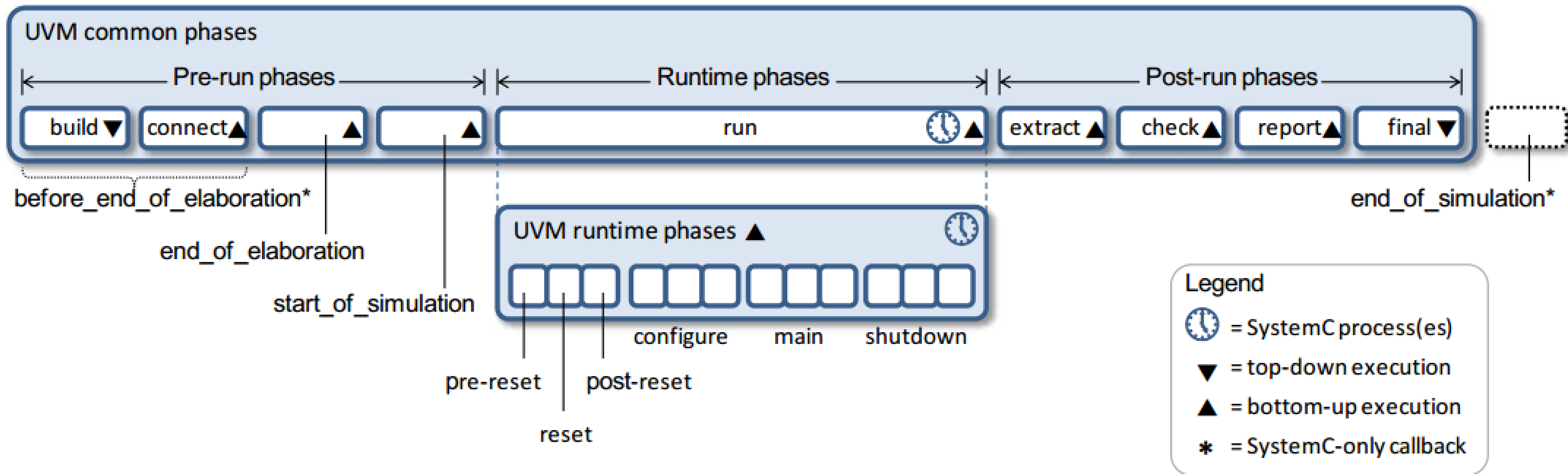
© Accellera Systems Initiative

Migrating to UVM-SystemC framework

- UVM-SystemC adheres to the UVM-SystemVerilog standard layered architecture
  - Migration of previous components to their respective layers required



UVM-SystemC Phasing



- UVM phases are mapped to the System-C phases
- Completion of a runtime phase happens as soon as there are no objections (anymore) to proceed to the next phase.
- All UVM components have the phases associated with them

Build flow of UVM-SystemC(1/2)

- GCC used for compiling UVM-SystemC framework with SystemC DUT
- VCS used for compiling UVM-SystemC framework with RTL DUT
- `run_test("<test_name>");`
- Simulator looks for a component *registered* with the name `<test_name>`
- Executes the `build_phase` (from top-down) of all the uvm components
- Elaborates the hierarchy and understands the component connections
- Executes the `run_phases` of all the components (in parallel)
- Enters `post_run` phases and finished once all objections are dropped

accellera SYSTEMS INITIATIVE

© Accellera Systems Initiative

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Build flow of UVM-SystemC(2/2)

Starting SC tests ...
0 s: build_phase top_test
0 s: constructor top_env
0 s: build_phase top_test.top_env
0 s: constructor agent
0 s: build_phase top_test.top_env.agent
0 s: constructor ahb_sequencer_inst
0 s: constructor ahb_driver_inst
0 s: build_phase top_test.top_env.agent.ahb_monitor_inst
0 s: connect_phase top_test.top_env.agent
0 s: connect_phase top_test.top_env
in ahb_reset_proc begin
0 s: UVM test with ahb_wr_rd_seq started top_test
in ahb_reset_proc end
100 ns: UVM test with ahb_wr_rd_seq started top_test
………..
1732630 ns: UVM test with ahb_wr_rd_seq finished top_test

--- UVM Report Summary ---
** Report counts by severity
UVM_ERROR : 0
UVM_FATAL : 0
UVM_INFO : 0
UVM_WARNING : 0
** Report counts by id
[RNTST] 1
[agent] 1
[ahb_driver_inst] 1705
[ahb_monitor_inst] 19
[ahb_seq] 2609
[pp_dynamic_cfg_seq] 8

Transaction Layer

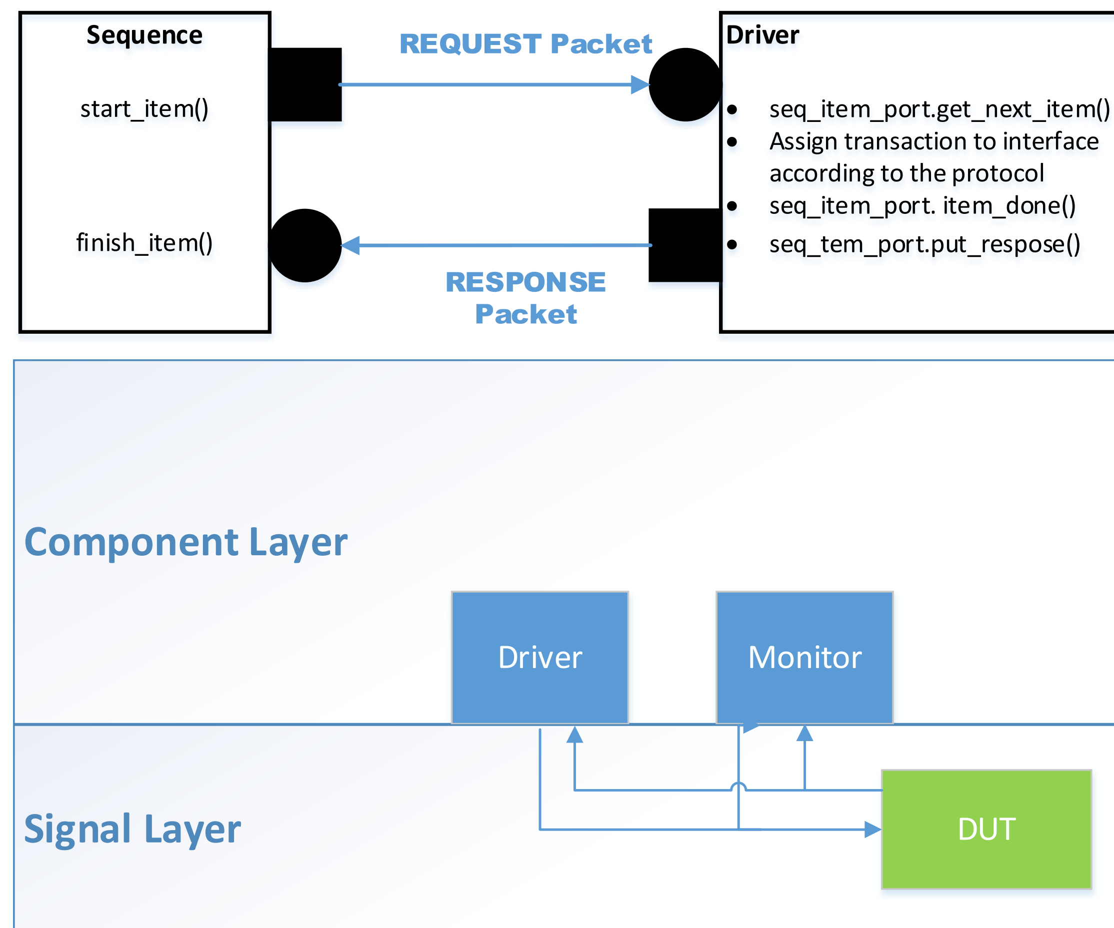
- Create interface and transaction classes as needed by the protocol
- Connect DUT to the interface
- Pass this interface to other components throughout the testbench hierarchy

```
int sc_main(int, char*[]) {
    ahb_clk_reset_gen* clk_rst_gen
        = new ahb_clk_reset_gen("clk_rst_gen");

    ahb_if* dut_if_in = new ahb_if("dut_if_in");
    dut_if_in->hclk(clk_rst_gen->ahb_clk);
    dut_if_in->hresetn(clk_rst_gen->reset_val);

    dut ahb_dut("ahb_dut");
    ahb_dut.hclk(clk_rst_gen->ahb_clk);
    ahb_dut.hresetn(clk_rst_gen->reset_val);
    ahb_dut.haddr(dut_if_in->haddr);
    ...

    uvm::uvm_config_db<ahb_if*>::
      set(0, "*", "vif", dut_if_in);
    uvm::uvm_config_db<sc_event*>::
      set(0, "*", "reset_done", clk_rst_gen->reset_done);
    run_test("ahb_wr_rd_test");
    return 0;
}
```


© Accellera Systems Initiative

Component Layer

- All components in this layer, are mapped to a transaction type
  - The transaction class constitutes on the packet type which is being transmitted across components
- Driver-Sequencer to follow a standard handshaking interface as per UVM standard
- Driver is the key component where all protocol intelligence has to be implemented
- Monitor can implement protocol checks, data integrity checks etc.
  - Taps the DUT signal directly



© Accellera Systems Initiative

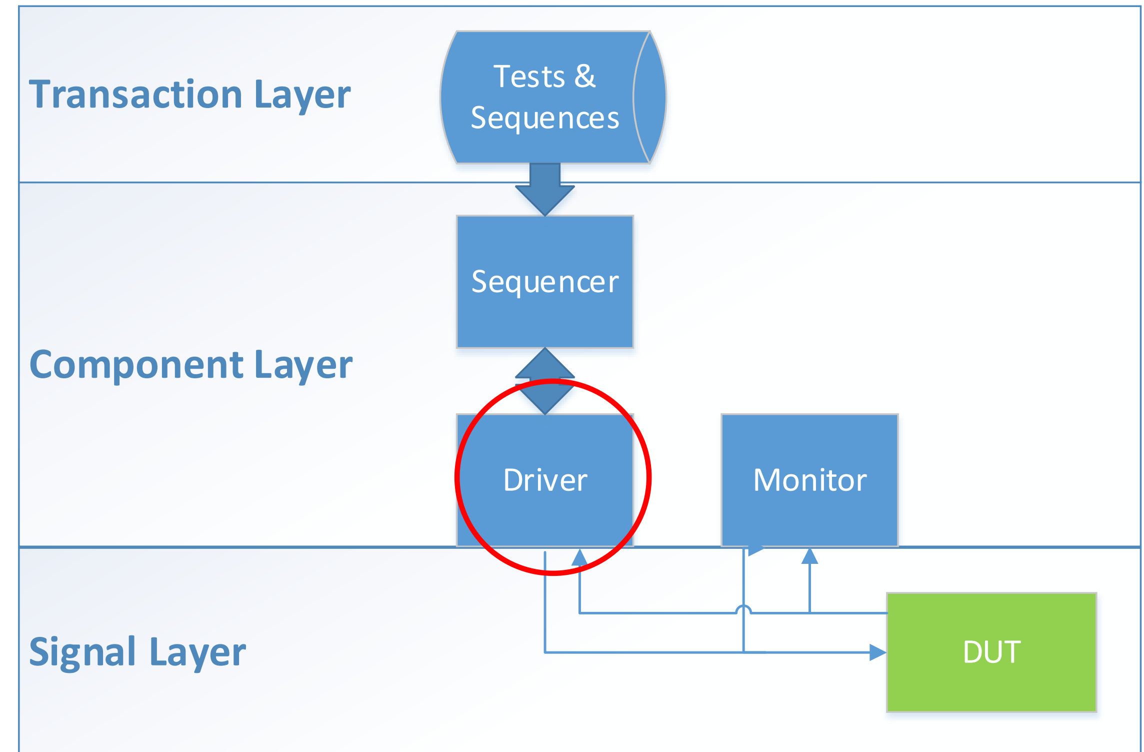
AHB Driver Component(1/6)

- Driver is derived from `uvm_component` base class
  - `uvm_component`s are created statically during the simulation, unlike `uvm_objects`
  - Module registered to factory by using "**UVM_COMPONENT_PARAM_UTILS**"
    - Enables component overrides using factory
  - `build_phase`() "creates" the component and gets the virtual interface handle
    - Virtual interface is the mode of communication between the DUT and the UVM_COMPONENTs
    - Interface contains variables which are to be passed across the components.
    - These contain all of the *transaction object* variables and some additional sideband signals, if any (based on the protocol of the driver)

© Accellera Systems Initiative

AHB Driver Component(2/6)

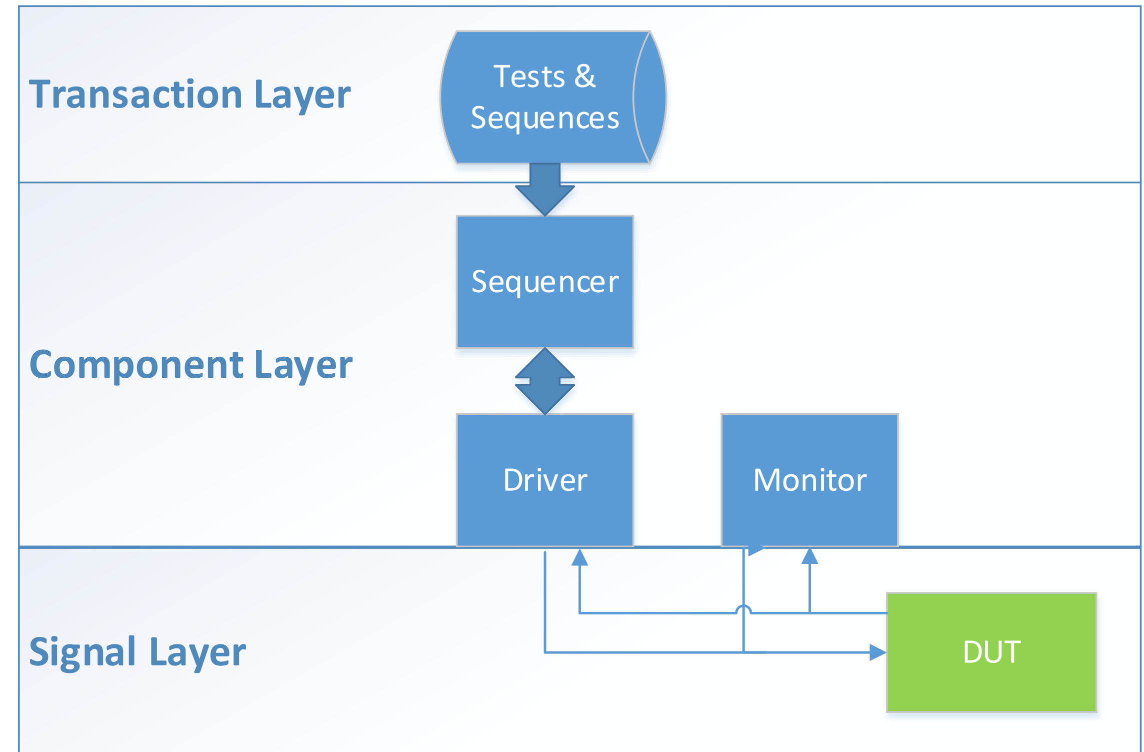
```
class ahb_driver: public
    uvm::uvm_driver<ahb_transaction> {

  public:
    UVM_COMPONENT_PARAM_UTILS(ahb_driver);
    ahb_if* ahb_vif;
    sc_event* reset_event_driver;
    sc_semaphore ahb_pipeline_lock;

    ahb_driver(
      uvm::uvm_component_name name
        = "ahb_driver“
    ):uvm::uvm_driver<ahb_transaction>(name),
      ahb_pipeline_lock(1)
    {
      ...
    }
```



© Accellera Systems Initiative

AHB Driver Component(3/6)

```
void build_phase(uvm::uvm_phase& phase) {

  UVM_INFO(this->get_name(),"build_phase entered",UVM_LOW);

  uvm_driver<ahb_transaction>::build_phase(phase);
  reset_event_driver = new sc_event("reset_event_driver");


  if (!uvm_config_db<ahb_if*>::
    get(this, "*", "vif", ahb_vif))
  {
    UVM_FATAL(this>get_name(),
      "AHB Virtual Interface missing");
  }

  if (!uvm_config_db<sc_event*>::
    get(this, "*", "reset_done", reset_event_driver))
  {
    UVM_FATAL(this->get_name(), "Reset event missing");
  }

}
```



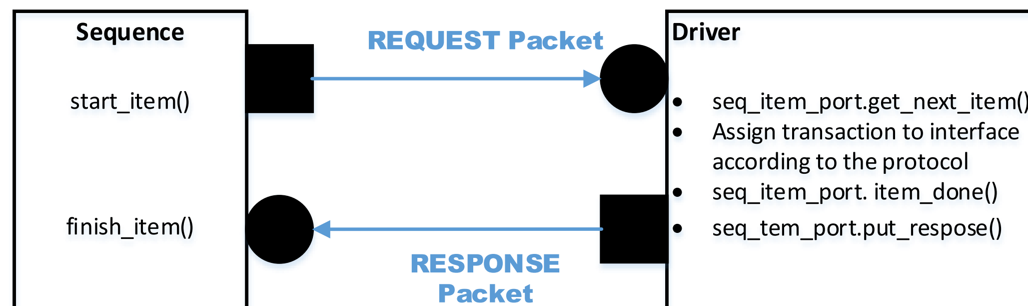
AHB Driver Component(4/6)

- Run_phase()
  - This is the phase where simulation time advances
  - This phase is supposed to handle the pre-reset and post-reset behavior of the driver
  - The variables in the virtual interface are assigned values based on various modes of operation of the driver (as per the protocol)
  - All the DUT interface signals which are to be driven by the testbench, should be assigned in this phase
  - Driver receives the transaction object from the sequencer
    - The communication (between driver and sequencer/sequence) is as per a UVM standard protocol

© Accellera Systems Initiative

Driver-Sequence Interactions

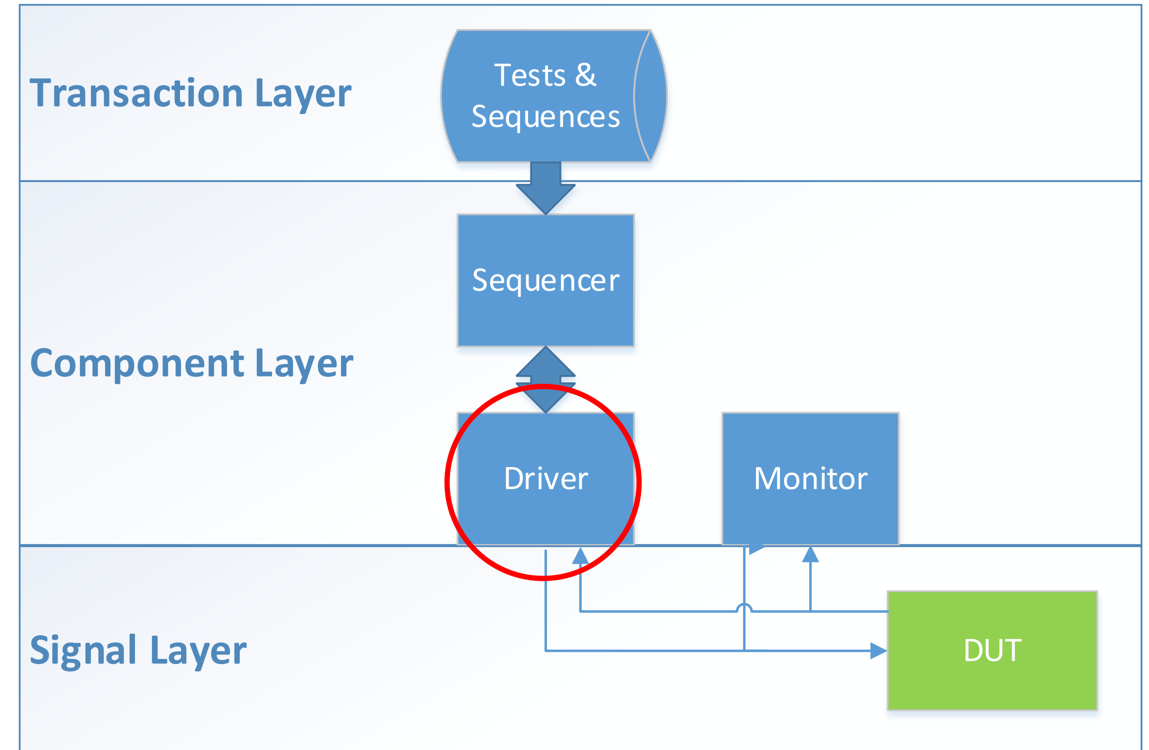
- Driver waits for a transaction item in the `run_phase()` by calling `get_next_item()` method of the driver analysis port.
- Once the sequence is started (from the uvm test), the `start_item()` method will be called and a transaction item reference will be passed
- Driver receives this and assigns relevant values to the sequence item
- `Item_done()` method call in driver indicates the transaction is updated and ready to be sent back to the sequence
- Driver can choose to call `put_response()` method and send a response packet with updated response fields (like status, data etc.)
- Sequence finished the item processing once the response is received and calls `finish_item()`



accellera SYSTEMS INITIATIVE
2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

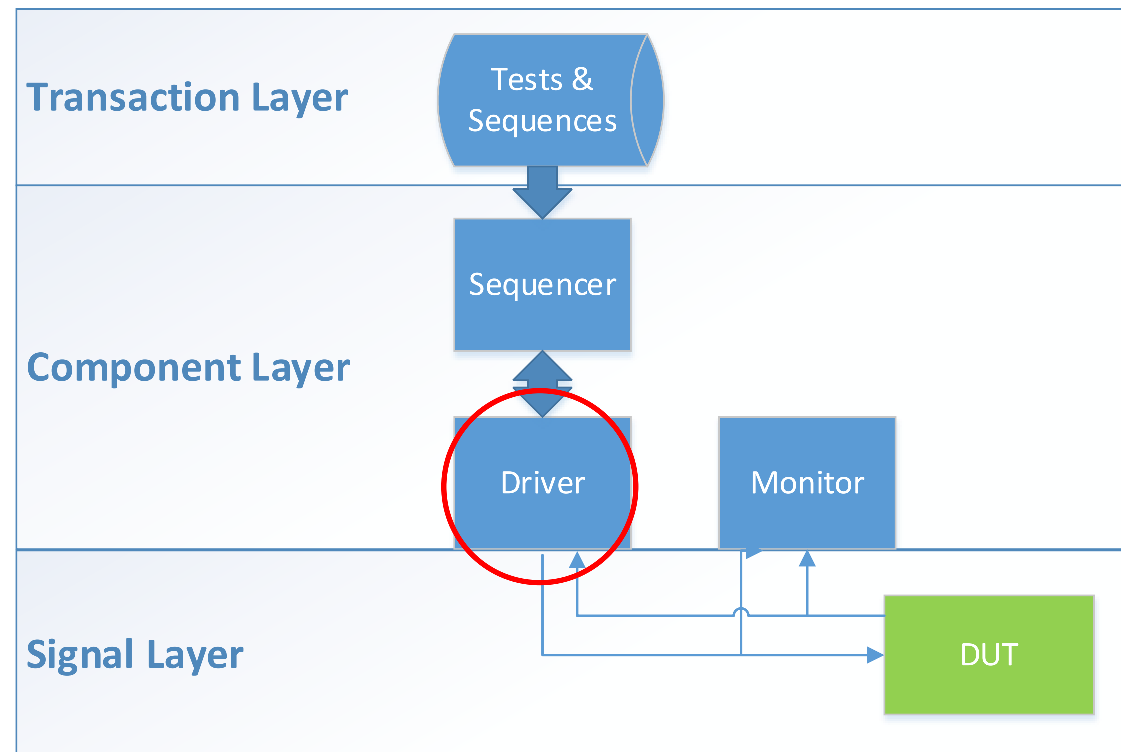
AHB Driver Component(5/6)

```
void run_phase(uvm::uvm_phase& phase) {
  UVM_INFO(this->get_name(),
    "run_phase entered",UVM_LOW);

  if (ahb_vif->hresetn == 0) wait(*reset_event_driver);

  while(true) {
    SC_FORK
    sc_spawn(sc_bind(
      &ahb_driver::send_transaction,this),"drive1"),
    sc_spawn(sc_bind(
      &ahb_driver::send_transaction,this),"drive2")
    SC_JOIN
  }
  UVM_INFO(this->get_name(),
    "run_phase finished", UVM_LOW);
}
```



AHB Driver Component(6/6)

```
void send_transaction() {
  ahb_transaction req, rsp;
  ahb_pipeline_lock.wait();
  UVM_INFO(this->get_name(),
    "send_transaction: next item",UVM_LOW);
  this->seq_item_port->get_next_item(req);
  ahb_vif->htrans      = req.htrans;
  ahb_vif->haddr       = req.haddr;
  ahb_vif->hsize       = req.hsize;

  ...

  wait(AHB_CLK);
  while (ahb_vif->hready != 1) wait(AHB_CLK);
  rsp.set_id_info(req);
  this->seq_item_port->item_done();
  this->seq_item_port->put_response(rsp);
  ahb_pipeline_lock.post();
}
```

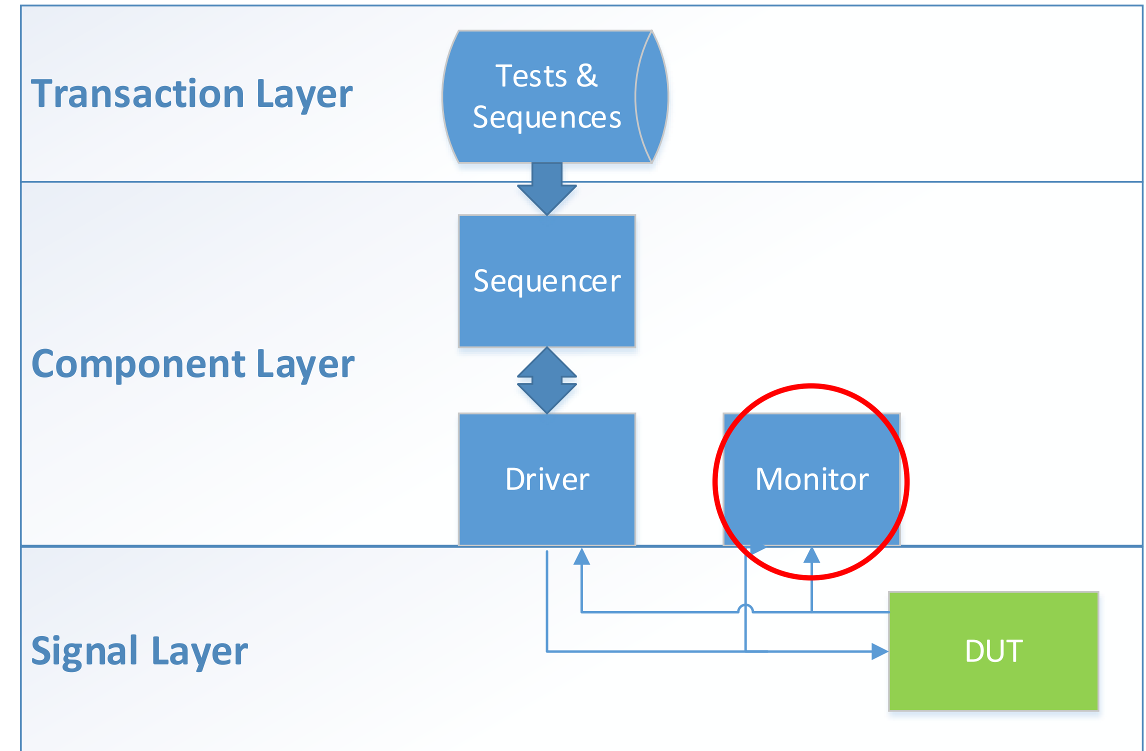


AHB Monitor Component(1/3)

- Monitor is derived from `uvm_monitor/uvm_component` base class
- Consist of an analysis port
- `build_phase()` "creates" the component and gets the virtual interface handle
- `run_phase()`
  - Monitor receives the transaction items through the virtual interface and creates an internal packet from it
  - If a broadcast packet is needed, monitor can write this packet to an analysis port and any other components can receive this by connecting to this port
  - Protocol checks and assertions are implemented to validate the transaction item values
  - Fatal/Error/Warning messages can be flagged based on the protocol failure severity

AHB Monitor Component(2/3)

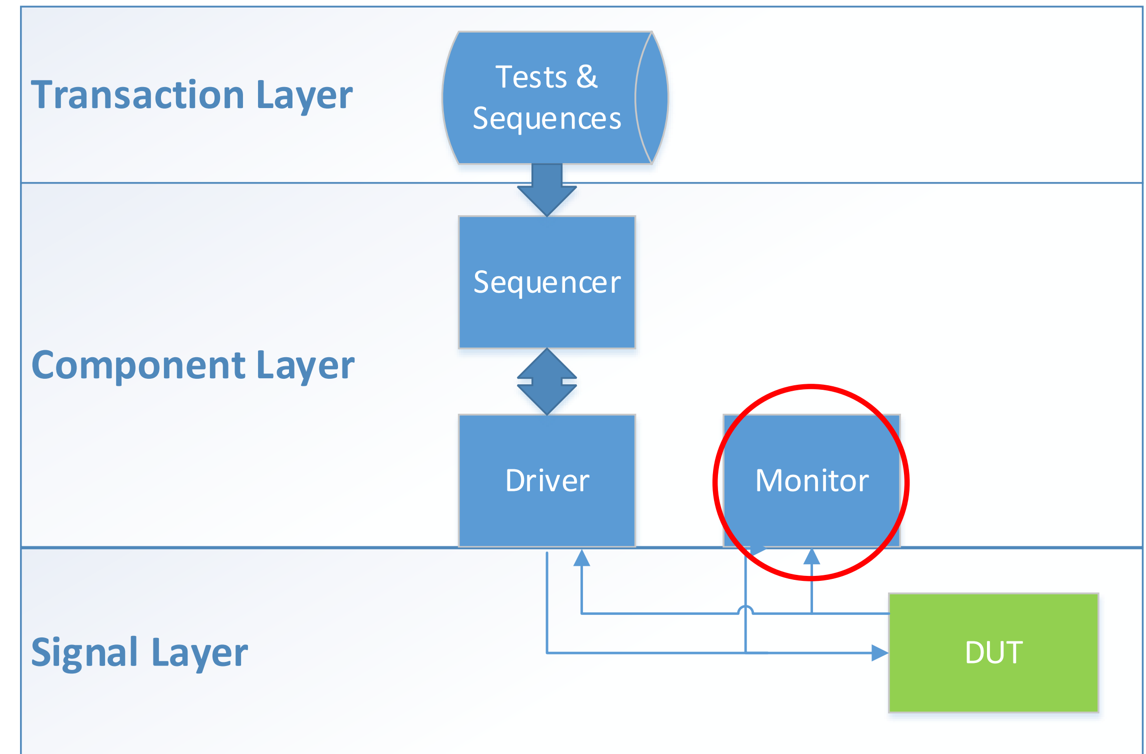
```
class ahb_monitor : public uvm::uvm_monitor {
  public:
    uvm::uvm_analysis_port<ahb_transaction>
      item_collected_port;
    ahb_if* vif;
    ahb_monitor(
      uvm::uvm_component_name name = "ahb_monitor"):
      uvm_monitor(name),
      item_collected_port("item_collected_port"),
      vif(0)
    { ... }

    UVM_COMPONENT_UTILS(ahb_monitor);

    void build_phase(uvm::uvm_phase& phase) {
      uvm::uvm_monitor::build_phase(phase);
      if (!uvm::uvm_config_db<ahb_if*>::
        get(this, "*", "vif", vif)) {
          UVM_FATAL(name(),
            "Virtual interface not defined!");
      }
    }
```



© Accellera Systems Initiative

AHB Monitor Component(3/3)

```
void run_phase( uvm::uvm_phase& phase ) {

  ahb_transaction pkt;

  while (true) { // monitor forever
    std::ostringstream str;
    wait( vif->hresetn.posedge_event());

    if (vif->hclk == 0)
      sc_core::wait(vif->hclk.posedge_event());

    pkt.htrans      = vif->htrans;
    pkt.haddr       = vif->haddr;
    ...
    item_collected_port.write(pkt);

    // Checks on the packet items
    AddressByteAlligned(pkt.haddr);
    SlaveErrorResponse(pkt);
  }
```



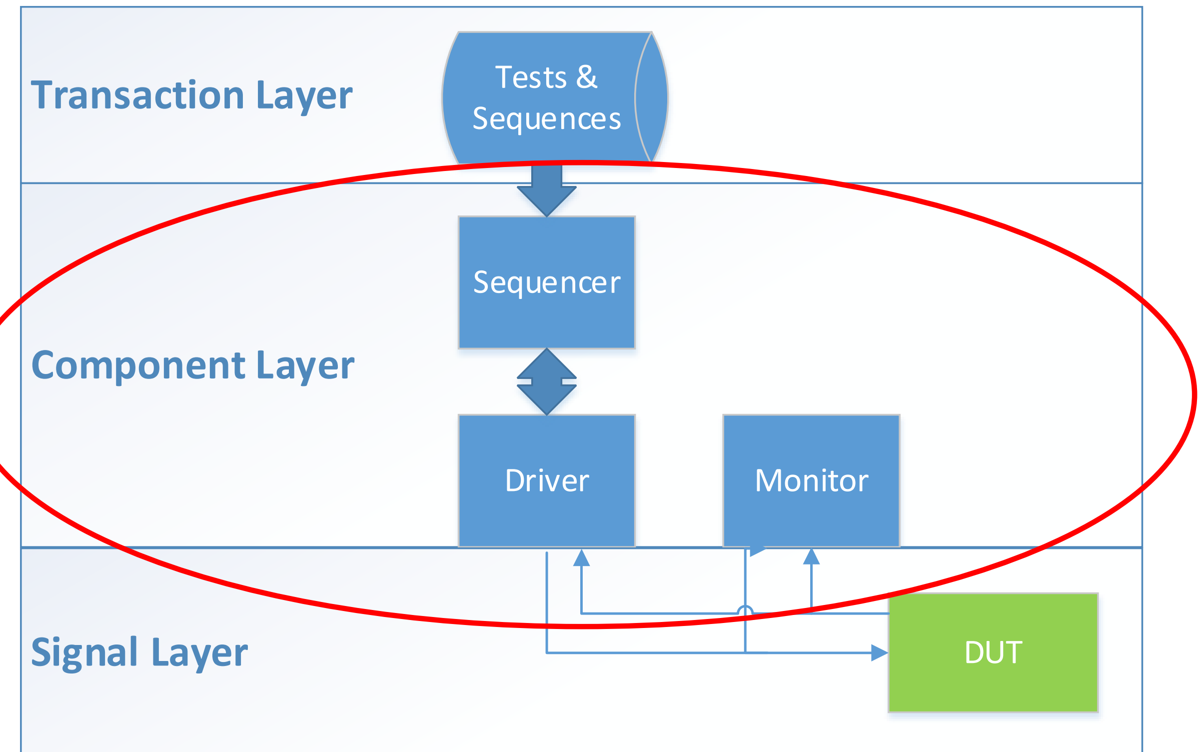
accellera SYSTEMS INITIATIVE

© Accellera Systems Initiative

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

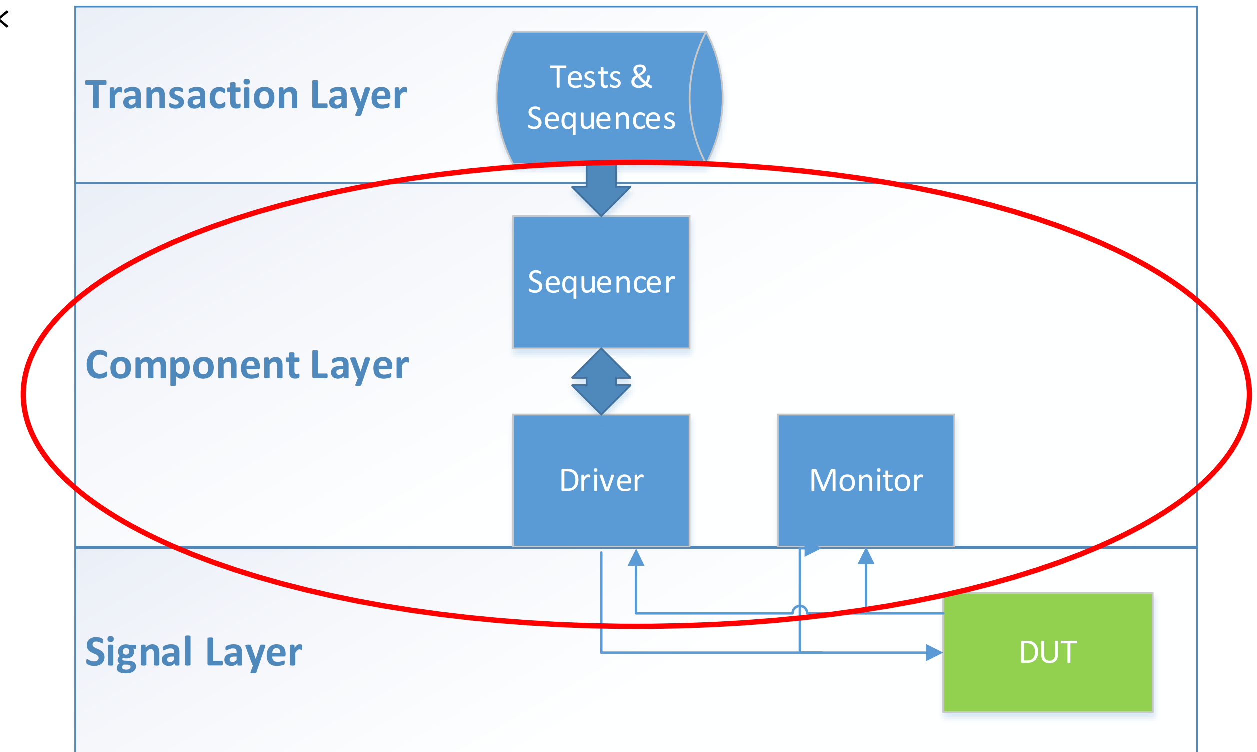
AHB Agent (1/3)

```
class ahb_agent : public uvm::uvm_agent {
  public:
    ahb_driver* ahb_driver_inst;
    ahb_monitor* ahb_monitor_inst;
    ahb_sequencer<ahb_transaction>* ahb_sequencer_inst;
    uvm::uvm_analysis_port<ahb_transaction>
      agent_item_collected_port;

    ahb_agent(uvm::uvm_component_name name = "ahb_agent"):
      uvm_agent(name), ahb_sequencer_inst(0),
      ahb_driver_inst(0) , ahb_monitor_inst(0),
      agent_item_collected_port("agent_item_collected_port")
    {
      std::cout << sc_core::sc_time_stamp()
        << ": constructor " << name << std::endl;
    }
…
```



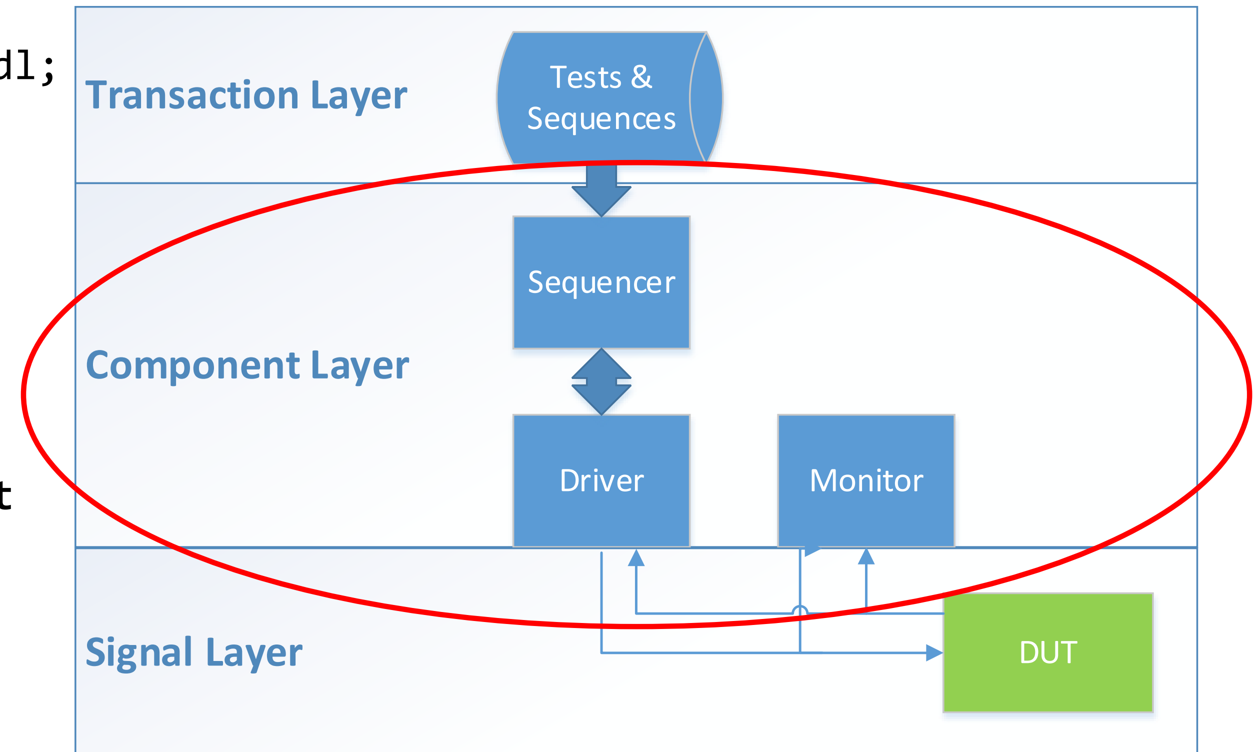
AHB Agent (2/3)

```
void build_phase(uvm::uvm_phase& phase) {

  std::cout << sc_core::sc_time_stamp() << ": build_phase " <<
    name() << std::endl;
 if (get_is_active() == uvm::UVM_ACTIVE) {
    UVM_INFO(this->get_name(), " is ACTIVE", UVM_LOW);

    ahb_sequencer_inst = ahb_sequencer<ahb_transaction>::
      type_id::create("ahb_sequencer_inst",this);

    ahb_driver_inst = ahb_driver::
      type_id::create("ahb_driver_inst",this);

    ahb_monitor_inst =
      ahb_monitor::type_id::create("ahb_monitor_inst",this);
  } else {
    ahb_monitor_inst = ahb_monitor::type_id::
      create("ahb_monitor_inst",this);

  UVM_INFO(this->get_name(), " is PASSIVE", UVM_LOW);
}
```



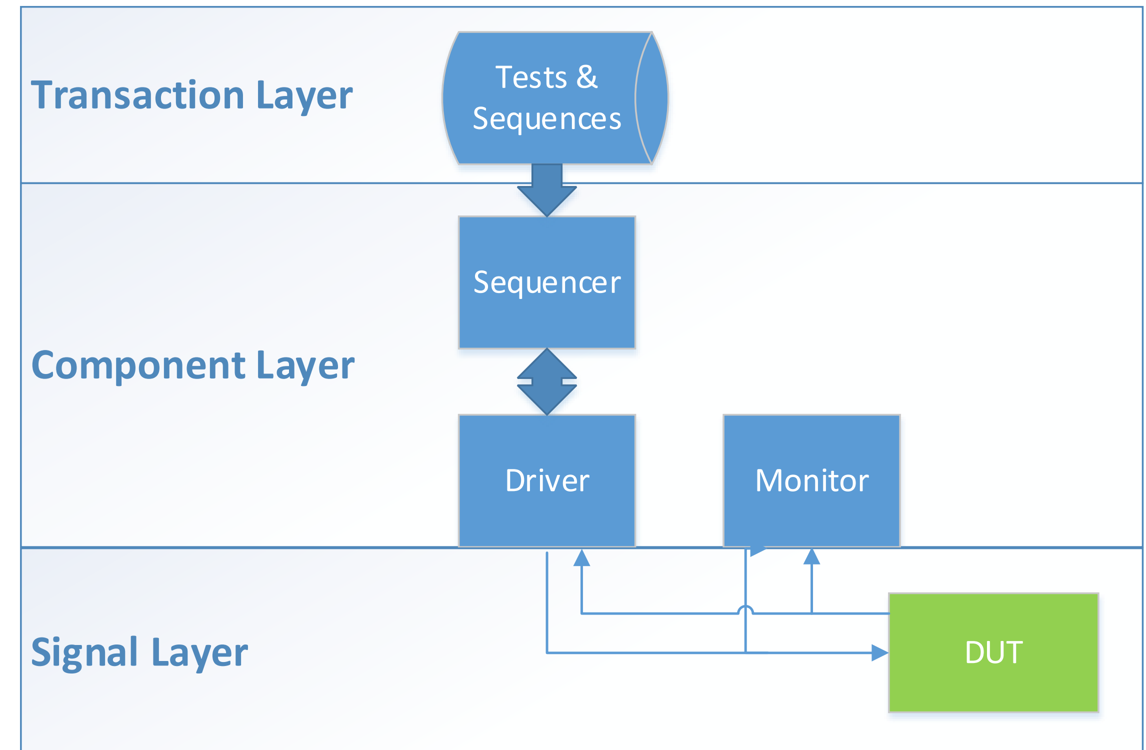
AHB Agent (3/3)

```
void connect_phase(uvm::uvm_phase& phase) {
  std::cout << sc_core::sc_time_stamp()
    << ": connect_phase " << name() << std::endl;

  if (get_is_active()==uvm::UVM_ACTIVE) {
    ahb_driver_inst->seq_item_port.connect
    (ahb_sequencer_inst->seq_item_export);
  }

  ahb_monitor_inst->item_collected_port.connect
  (agent_item_collected_port);
}
```



AHB Basic Env

```
class ahb_basic_env : public uvm::uvm_env {
  public:
    UVM_COMPONENT_UTILS(ahb_basic_env);
    ahb_agent* agent;
    ahb_basic_env(
      uvm::uvm_component_name name=
      "ahb_basic_env“): uvm::uvm_env(name), agent(0) {
      std::cout << sc_core::sc_time_stamp()
      << ": constructor " << name << std::endl;
    }

    void build_phase(uvm::uvm_phase& phase) {
      std::cout << sc_core::sc_time_stamp()
      << ": build_phase " << name() << std::endl;
      agent = ahb_agent::type_id::create("agent", this);

      uvm::uvm_config_db<int>::
      set(this, "agent", "is_active", uvm::UVM_ACTIVE);
    }

};
```



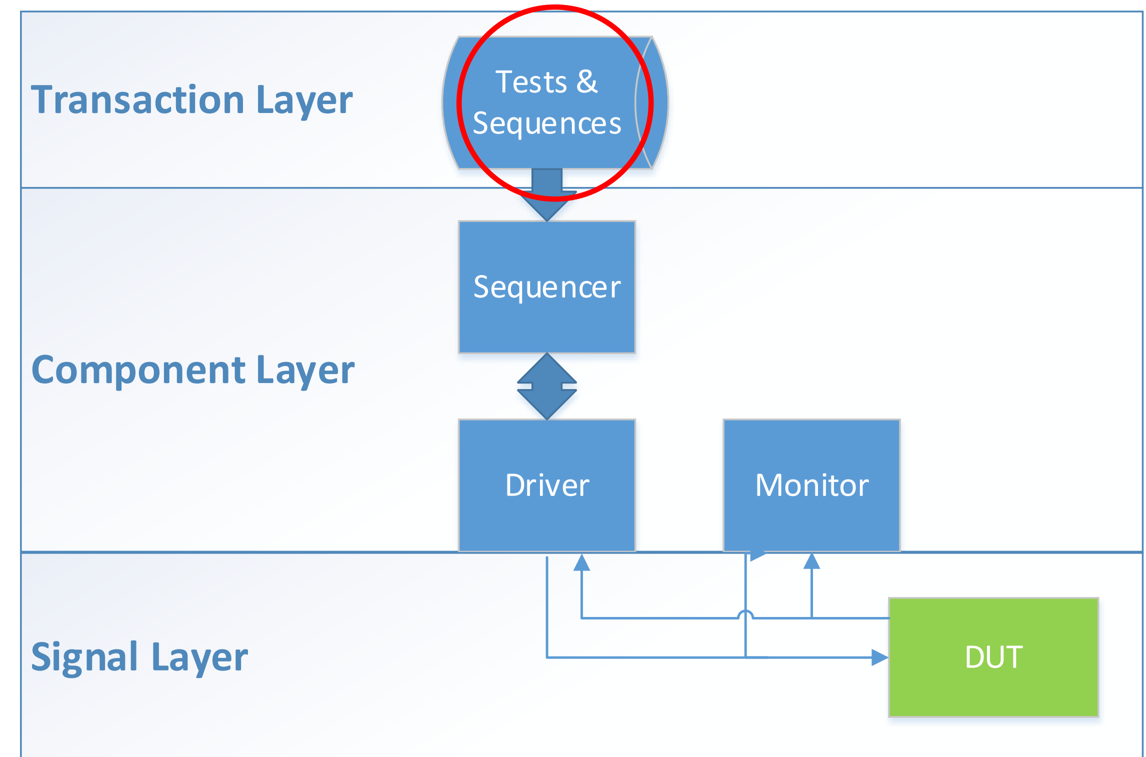
Transaction Layer (UVM Tests) (1/3)

- UVM_TESTs are responsible for building the top level environment and initiating the start of the required sequence
- UVM runs tasks on objections and all the components wishing to perform a task are expected to raise an objection
- `build_phase()` "creates" the component
- `run_phase()`
  - Objection is raised and dropped in this phase
  - Sequence handle is created and sequence is started by calling the `start()` method
  - The sequencer on which the sequence should be run is also specified
  - Multiple sequences can be started at the same time on different sequencers

© Accellera Systems Initiative

Transaction Layer (UVM Tests) (2/3)

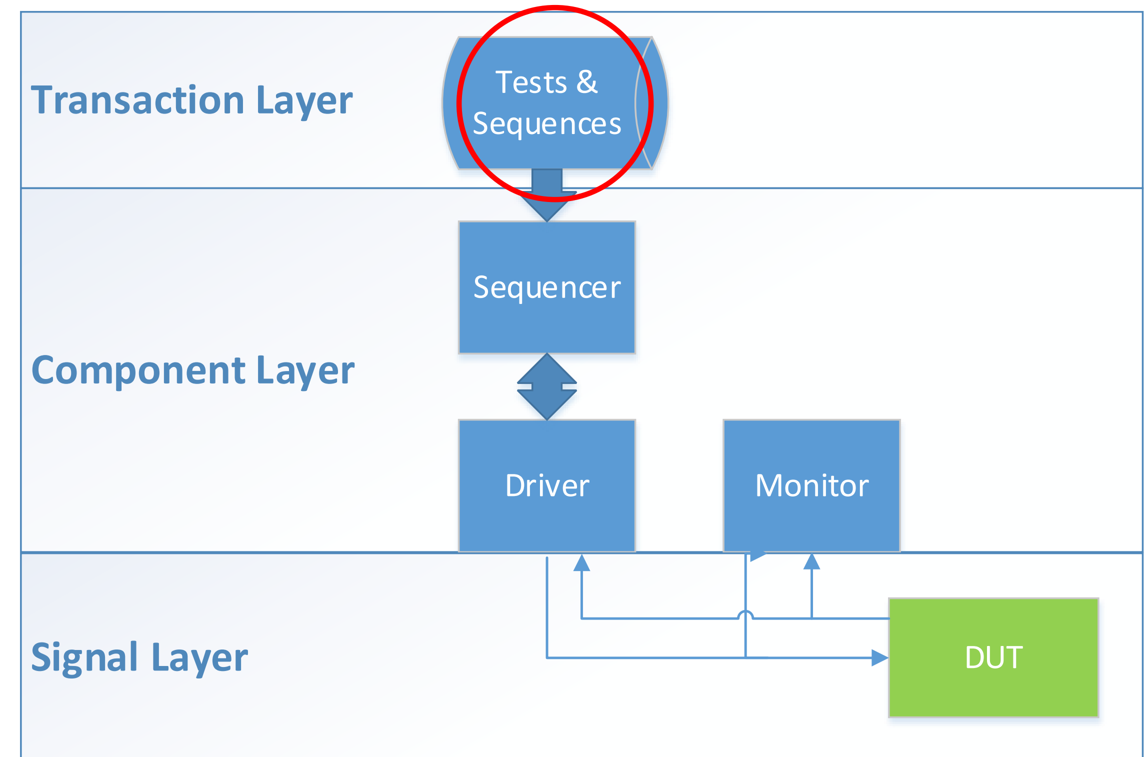
```
class ahb_wr_rd_test : public uvm::uvm_test {
 public:
   ahb_wr_rd_sequence* ahb_wr_rd_seq;
   ahb_basic_env* top_env;

   UVM_COMPONENT_UTILS(ahb_wr_rd_test);
   ahb_wr_rd_test( uvm::uvm_component_name
   name = "ahb_wr_rd_test"):
   uvm::uvm_test( name ), top_env(0) {}

   virtual void build_phase(uvm::uvm_phase& phase){
     std::cout << sc_core::sc_time_stamp()
     << ": build_phase " << name() << std::endl;
     uvm_test::build_phase(phase);
     top_env =  ahb_basic_env::type_id::
                create("top_env",this);}
```



© Accellera Systems Initiative

Transaction Layer (UVM Tests) (3/3)

```
virtual void run_phase(uvm::uvm_phase& phase) {
      std::cout << sc_core::sc_time_stamp()
      << ": UVM test with ahb_wr_rd_seq started "
      << name() << std::endl;

      phase.raise_objection(this);

      ahb_wr_rd_seq =
      new  ahb_wr_rd_sequence("ahb_wr_rd_seq");
      ahb_wr_rd_seq->start(top_env->
      agent->ahb_sequencer_inst);

      phase.drop_objection(this);
      std::cout << sc_core::sc_time_stamp()
      << "UVM test with ahb_wr_rd_seq finished"
      << name() << std::endl;
}
```



© Accellera Systems Initiative

PART 2: RANDOMIZATION USING SCV & CRAVE

© Accellera Systems Initiative

Overview

- Sequence randomization
  - SCV
  - CRAVE
- Sequence randomization sample

Random Sequences Using SCV (1/2)

- While using SCV for randomizing the sequence item, `scv_extensions` are to be created
- `SCV_EXTENSION` consist of the transaction fields to be randomized
- `SCV_CONSTRAINTS` can be added for all of the `scv_extension` variables as per the required constrained randomization
- In a sequence, the `scv_constraint` object is created and `next()` method is called to get a set of random values
- These are assigned to the `scv_smart_ptr` for the transaction class
- Multiple random values can be received by calling `next()` method each time

accellera SYSTEMS INITIATIVE

© Accellera Systems Initiative

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Random Sequences Using SCV (2/2)

Create `scv_extensions` for the sequence item class i.e. for the transaction type

```
SCV_EXTENSIONS(ahb_transaction) {
  public:
    scv_extensions<sc_uint<ahbConfig::AhbAddrWidth>>   haddr;
    scv_extensions<sc_uint<ahbConfig::AhbDataWidth>[BURSTLENGTH]>
                                                       hwdata;
    scv_extensions< sc_uint<ahbConfig::AhbBurstSize>>  hburst;
    scv_extensions< sc_uint<ahbConfig::AhbSize>>       hsize;

    SCV_EXTENSIONS_CTOR(ahb_transaction) {
      SCV_FIELD(haddr);
      SCV_FIELD(hburst);
      SCV_FIELD(hsize);
      SCV_FIELD(hwdata);
    }

    bool has_valid_extensions() { return true; }
};
```

Create constraints class using `smart_ptr` of sequence item class type

```
class ahb_trans_constraints : virtual public scv_constraint_base {
  public:
    scv_smart_ptr<ahb_transaction> req;
    SCV_CONSTRAINT_CTOR(ahb_trans_constraints) {
      SCV_CONSTRAINT((req->haddr() * 0x3) == 0x0);
      SCV_CONSTRAINT(
        (req->hburst() >= ahbConfig::HBURST_SINGLE) &&
        (req->hburst() <= ahbConfig::HBURST_INCR16)
      );
      SCV_CONSTRAINT(
        (req->hsize() >= ahbConfig::HSIZE_BYTE) &&
        (req->hsize() <= ahbConfig::HSIZE_WORD)
      );

      // For wrapping bursts, start address from an address
      // other that 0x00 offset
      SCV_CONSTRAINT(
        if_then(req->hburst() == ahbConfig::HBURST_WRAP4,
          ((req->haddr() * 0x7) != 0x0) );
      );
    }
```


accellera SYSTEMS INITIATIVE
2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Hierarchical Sequences using Random Sequence Items (SCV)(1/2)

```
class ahb_wr_rd_sequence : public
uvm::uvm_sequence<ahb_transaction>
{
    public:
        UVM_OBJECT_UTILS(ahb_wr_rd_sequence);

UVM_DECLARE_P_SEQUENCER(ahb_sequencer<ahb_transaction>);
        ahb_if* ahb_vif_seq;
        ahb_wr_rd_sequence( const std::string&
        name = "ahb_wr_rd_sequence") :

uvm::uvm_sequence<ahb_transaction> ( name ){}
        uint8_t xactType;
        unsigned addrValue;
        unsigned dataValue;
        void body()
        {
            UVM_INFO(this->get_name(), "Starting sequence",
uvm::UVM_INFO);
            ahb_trans_constraints constr_req("constr_req");
            scv_smart_ptr<ahb_transaction>
                 rand_smart_ptr_ahb_pkt;
            ahb_basic_sequence* ahb_seq;
            ahb_seq = new ahb_basic_sequence("ahb_seq");
```

```
constr_req.next();
rand_smart_ptr_ahb_pkt.write(constr_req.req.read());

ahb_seq->xactType = rand_smart_ptr_ahb_pkt ->hwrite;
ahb_seq->hburstValue =
                   rand_smart_ptr_ahb_pkt ->hburst;
ahb_seq->addrValue = rand_smart_ptr_ahb_pkt ->haddr;
ahb_seq->dataValue = 0xabababab;

ahb_seq->start(m_sequencer);

constr_req.next();
rand_smart_ptr_ahb_pkt.write(constr_req.req.read());

ahb_seq->xactType = rand_smart_ptr_ahb_pkt ->hwrite;
ahb_seq->hburstValue =
                    rand_smart_ptr_ahb_pkt ->hburst;
ahb_seq->addrValue = rand_smart_ptr_ahb_pkt ->haddr;

ahb_seq->start(m_sequencer);

UVM_INFO(this->get_name(),
         "Finishing sequence",  uvm::UVM_INFO);
```

Hierarchical Sequences using Random Sequence Items (SCV)(2/2)

```
class ahb_basic_sequence : public
uvm::uvm_sequence<ahb_transaction>
{
    public:
        UVM_OBJECT_UTILS(ahb_basic_sequence);
        uint8_t xactType;
        unsigned addrValue,dataValue;
        unsigned hburstValue, hsizeValue;
         ahb_basic_sequence( const std::string&
         name = "ahb_basic_sequence") :
uvm::uvm_sequence<ahb_transaction> ( name ) {}
        void body()
        {
           UVM_INFO(this->get_name(), "Starting
sequence ahb_basic_sequence", uvm::UVM_INFO)
            ahb_transaction* req_pkt;
            ahb_transaction* rsp;
            req_pkt = new ahb_transaction();
            rsp     = new ahb_transaction();
            single_wr_rd(addrValue,xactType,dataValue,
req_pkt, rsp);
        }
```

```
void single_wr_rd(unsigned addrValue,
                  unsigned xactType,
                  unsigned dataValue,
                  ahb_transaction* req_pkt,
                  ahb_transaction* rsp)
       {
           UVM_INFO(this->get_name(), "Initiating
non-burst accesses", uvm::UVM_INFO);

           req_pkt->haddr  = addrValue;
           req_pkt->hsel   = 1;
           req_pkt->hready = 1;
           req_pkt->htrans = ahbConfig::HTRANS_NONSEQ;
           req_pkt->hsize = hsizeValue;
           req_pkt->hwrite = xactType;
           req_pkt->hwdata[0] =
                          (sc_uint<32>)dataValue;

      ⇨    this->start_item(req_pkt);
           this->finish_item(req_pkt);
           this->get_response(rsp);
     }
```

Random Sequences Using CRAVE

- Transaction class need to be derived from `uvm_randomized_sequence_item()`
- Variables to be randomized are declared as `crv_variables`
- Constraints can be specified by using `crv_constraint` method
- base sequence using the transaction item, should call the `randomize()` method to get random values for the `crv_variables`
- values should be assigned to the transaction packets fields, as per requirement and sent to the DUT
- UVM_DO* macros can be called to specify which transaction object has to be sent to the driver and with what random values

Sequence Item Using CRAVE

```
class ahb_transaction : public uvm_randomized_sequence_item {
  public:
    UVM_OBJECT_UTILS(ahb_transaction);

    // define some rand variables
    crv_variable< sc_uint< ahbConfig::AhbAddrWidth > > haddr;
    crv_variable< sc_uint< ahbConfig::AhbSize >    >    hsize;
    crv_variable< sc_uint< ahbConfig::AhbDataWidth> >  hwdata[BURSTLENGTH];
    crv_variable< unsigned >                            hburst;

    // Add some constraints
    crv_constraint valid_hburst_range  {HBURST_SINGLE <= hburst() <= HBURST_INCR16};
    crv_constraint valid_hsize_range   {HSIZE_BYTE <= hburst() <= HSIZE_WORD};
    crv_constraint valid_addr_range    {haddr() * 0x3 == 0x0};
    crv_constraint addr_for_wrap_burst {if_then(hburst() == HBURST_WRAP4, (haddr() * 0x7) != 0x0)};

    // Constructor
    ahb_transaction(crv_object_name name = "ahb_transaction") : uvm_randomized_sequence_item(name) {
      ...
    };
};
```

Hierarchical Sequences using Random Sequence Item (CRAVE)

```
#include "ahb_basic_sequence.h"
class ahb_wr_rd_sequence : public uvm_randomized_sequence<ahb_transaction>
{
    public:

        UVM_OBJECT_UTILS(ahb_wr_rd_sequence);
        ahb_wr_rd_sequence( crave::crv_object_name name = "ahb_wr_rd_sequence") :
                            uvm_randomized_sequence<ahb_transaction> ( name )
        {
            cout << "Entered constructor of ahb_wr_rd_sequence " << endl;
        }

        void body()
        {
            UVM_INFO(this->get_name(), "Starting sequence", uvm::UVM_INFO);
            ahb_basic_sequence* ahb_seq;
            ahb_seq = new ahb_basic_sequence("ahb_seq");
            ahb_seq->hburstValue = ahbConfig::HBURST_SINGLE;
            ahb_seq->start(m_sequencer);
            UVM_INFO(this->get_name(), "Finishing sequence", uvm::UVM_INFO);
        }

};
```

Base sequence with crv_variable

```
class ahb_basic_sequence : public
uvm_randomized_sequence<ahb_transaction>
{
    public:
        UVM_OBJECT_UTILS(ahb_basic_sequence);
        crv_variable<uint8_t  >  xactType;
        crv_variable<unsigned >  addrValue;
        crv_variable<unsigned >  dataValue;
        ahb_basic_sequence( crave::crv_object_name
         name= "ahb_basic_sequence") :
         uvm_randomized_sequence<ahb_transaction> ( name )
         {}
        virtual ~ahb_basic_sequence() {
        };

        void body()
        {
            UVM_INFO(this->get_name(), "Starting sequence
ahb_basic_sequence", uvm::UVM_INFO);
            ahb_transaction* req_pkt;
            ahb_transaction* rsp;
            req_pkt = new ahb_transaction();
            rsp     = new ahb_transaction();
            single_wr_rd(addrValue,xactType,dataValue, req_pkt,
rsp);
        }
 }
```

```
void single_wr_rd(unsigned addrValue,
                  unsigned xactType,
                  unsigned dataValue,
                  ahb_transaction* req_pkt,
                  ahb_transaction* rsp)
{
    UVM_INFO(this->get_name(), "Initiating non-burst
accesses", uvm::UVM_INFO);
    this->randomize();
    req_pkt->haddr  = addrValue;
    req_pkt->hsel   = 1;
    req_pkt->hready = 1;
    req_pkt->htrans = ahbConfig::HTRANS_NONSEQ;
    req_pkt->hsize = hsizeValue;
    req_pkt->hwrite = xactType;
    UVM_DO_WITH(req_pkt, req_pkt->haddr() == addrValue);
    UVM_INFO(this->get_name(), "Exiting non-burst
accesses", uvm::UVM_INFO);
 }
```

accellera SYSTEMS INITIATIVE

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

SCV Sequence Randomization Sample

- SCV constraints written to configure the IP parameter randomly
- The IP is designed to find a path between point 'A' and 'B' without colliding to any obstacles on its path. Start, target and the obstacle map is an input to the IP.
- Test ends when an Interrupt is asserted by the IP; interrupt status of 1 => Valid output ready, interrupt status of 2 => No valid output(path) possible
- SCV library does not have *scv_extensions added for fixed point data types* yet.
- Hence, *constrained randomization attained using rand()* method.
- Sample plots show the capability of randomization to generate distinct scenarios.

accellera SYSTEMS INITIATIVE

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Code Snippet for Randomizing Fixed Point Variables(1/2)

```
    sc_fixed<32,8> SP_X, SP_Y, SP_PHI;
    sc_fixed<32,8> TP_X, TP_Y, TP_PHI;
    sc_fixed<32,8> omap_X, omap_Y;
    sc_fixed<32,8> rand_SP_PHI;
    sc_fixed<32,8> rand_TP_PHI;

    sc_fixed<32,8> eucDistObsSP;
    sc_fixed<32,8> eucDistObsTP;
    sc_fixed<32,8> eucDistSPTP;
    omap_count = (rand()%161) + 20; // obs points between 20
and 180

    SP_X = (sc_fixed<32,8>)(rand() / (RAND_MAX / 18.0) ) + (-
9.0);
    SP_Y = (sc_fixed<32,8>)(rand() / (RAND_MAX / 18.0) ) + (-
9.0);
    // Value between -1.5708 to 1.5708 i.e -90 to 90
    rand_SP_PHI = -1.5708 + (sc_fixed<32,8>)(1.5708 * (rand()
/ (RAND_MAX + (-1.5708))));
    rand_TP_PHI = -1.5708 + (sc_fixed<32,8>)(1.5708 * (rand()
/ (RAND_MAX + (-1.5708))));
```

```
 while(1)
  {
        TP_X = (sc_fixed<32,8>)(rand() / (RAND_MAX / 18.0) ) +
(-9.0);
        TP_Y = (sc_fixed<32,8>)(rand() / (RAND_MAX / 18.0) ) +
(-9.0);
        eucDistSPTP = sqrt((TP_X-SP_X)*(TP_X-SP_X) + (TP_Y-
SP_Y)*(TP_Y-SP_Y));
        if( (eucDistSPTP > 0.4) && (eucDistSPTP < 3) )
        {
             cout << "EP is " << TP_X << endl;
             ahb_seq->addrValue = TARGETPOSEX;
             wr_data.range(31,24) = TP_X.range(31,24);
             wr_data.range(23,0) =  TP_X.range(23,0);
             ahb_seq->dataValue = wr_data;
             ahb_seq->start(m_sequencer);
            break;
        }

  }
```

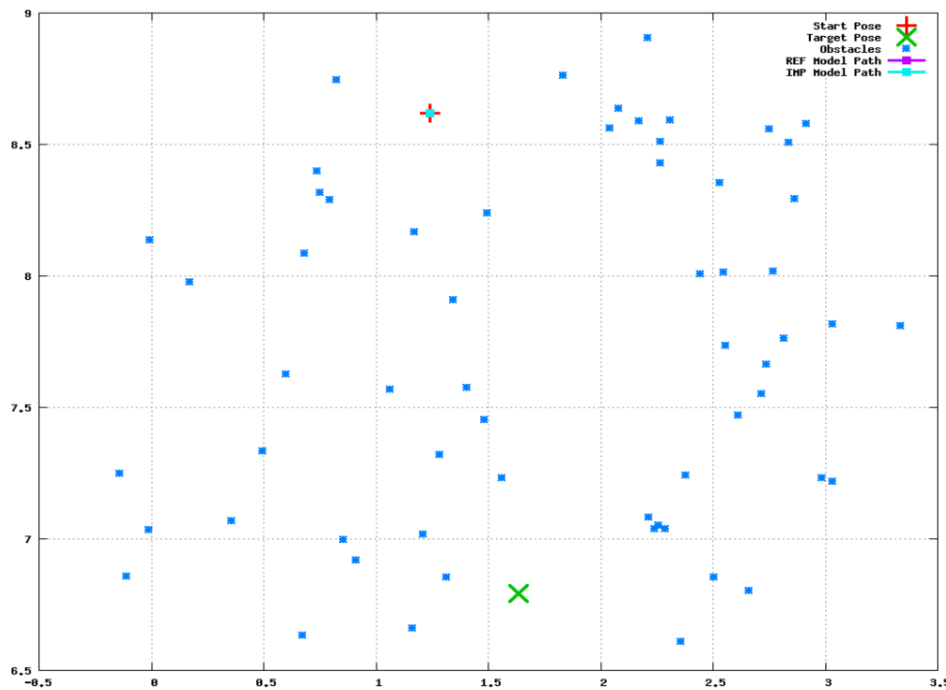
Code Snippet for Randomizing Fixed Point Variables(2/2)

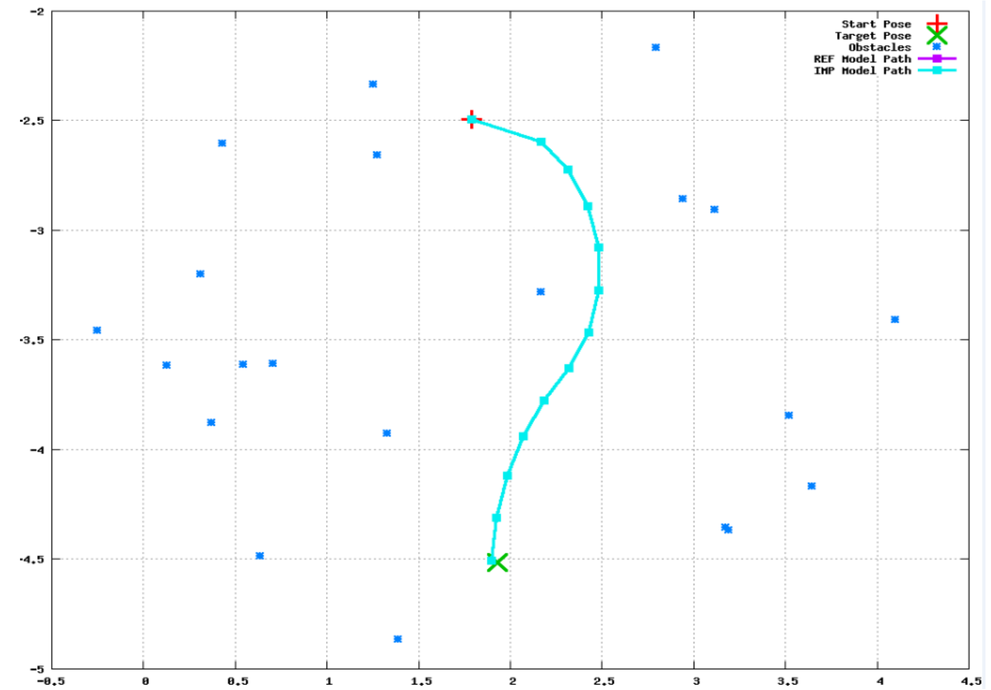
```
// Keep finding obst points for the required omap count.
Ignore points which are close to SP/TP
while(1)
{
     omap_X = (sc_fixed<32,8>)(rand() / (RAND_MAX / 18.0) ) + (-
9.0);
     omap_Y = (sc_fixed<32,8>)(rand() / (RAND_MAX / 18.0) ) + (-
9.0);

     // Calculate euc dist of obt point from SP and TP
     eucDistObsSP = sqrt((SP_X-omap_X)*(SP_X-omap_X) + (SP_Y-
omap_Y)*(SP_Y-omap_Y));
     eucDistObsTP = sqrt((TP_X-omap_X)*(TP_X-omap_X) + (TP_Y-
omap_Y)*(TP_Y-omap_Y));

     if( (eucDistObsSP > 0.3)
        && (eucDistObsSP < (1.25*eucDistSPTP) )
        && (eucDistObsTP > 0.3)
        && (eucDistObsTP < (1.25*eucDistSPTP) )) {
        act_omap_count++;
        omap_cfg << omap_X << " " << omap_Y;
     }
```

- Sample SCV for fixed point variables(not supported yet):

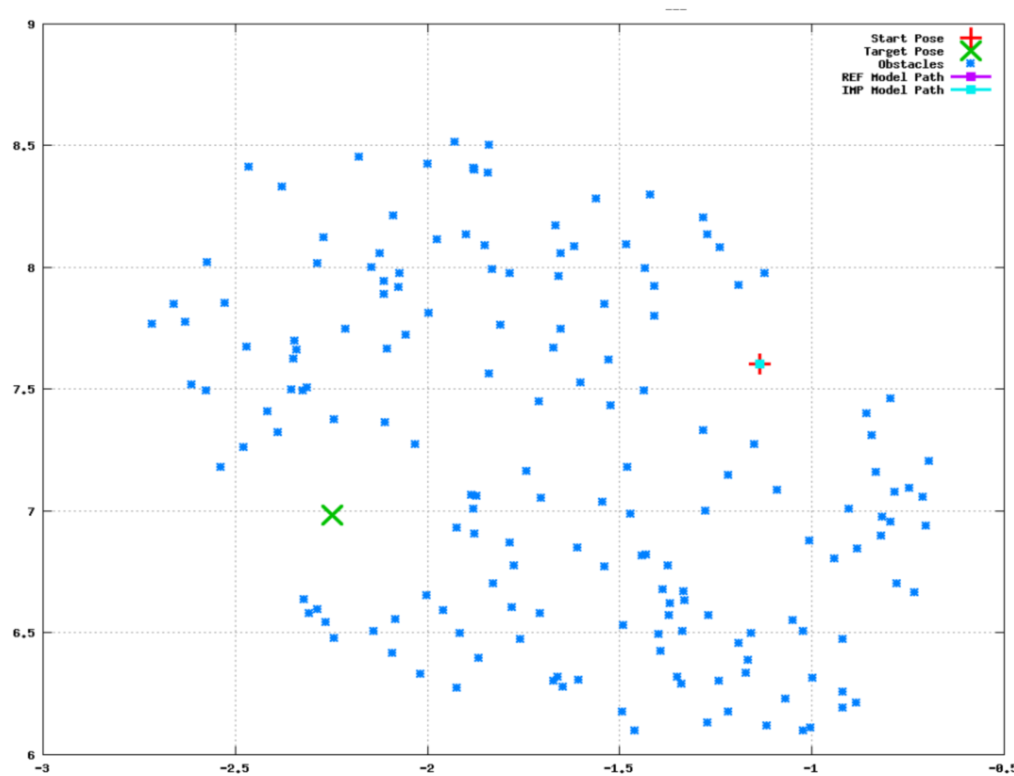
```
SCV_EXTENSIONS(PathPoints)
{
    public:
         scv_extensions<
sc_fixed<32,8,SC_DEFAULT_Q_MODE_,
SC_DEFAULT_O_MODE_,SC_DEFAULT_N_BITS_>>
targetPoseXY;

        SCV_EXTENSIONS_CTOR(PathPoints)
        {
            SCV_FIELD(targetPoseXY);
        }
        bool has_valid_extensions() {return
true;}
}
```

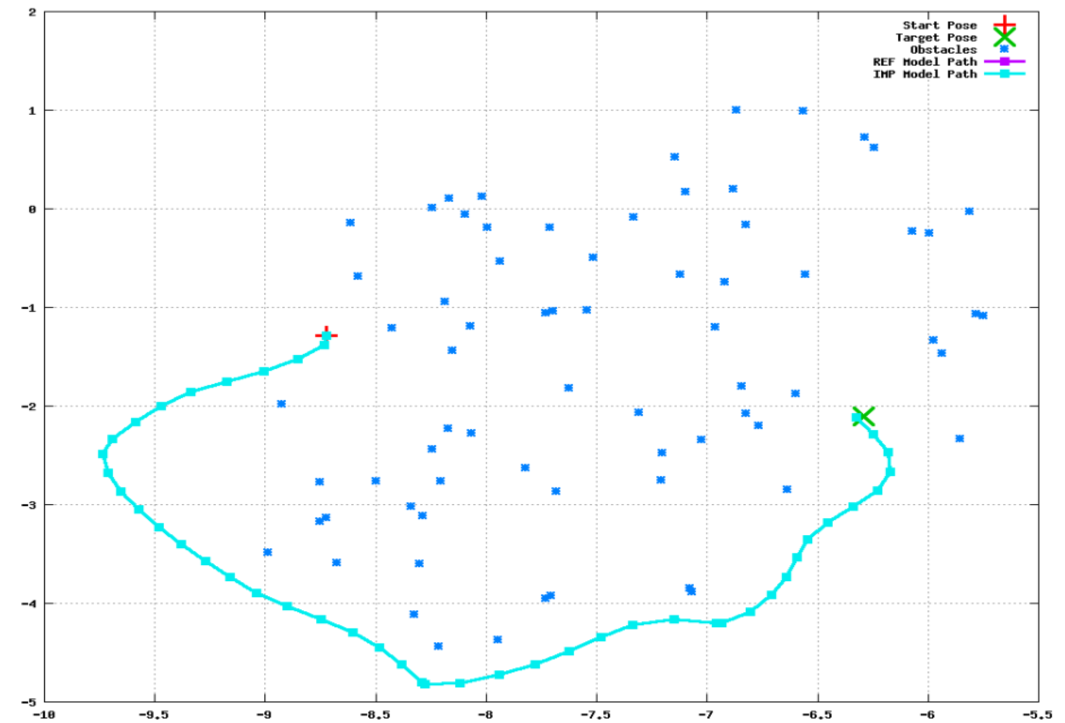
Sample IP outputs (1/2)





Sample IP outputs (2/2)





© Accellera Systems Initiative

PART 3: MIGRATION RESULTS & CONCLUSION

Overview

- Benefits
- Tooling
- Conclusion

© Accellera Systems Initiative

Potential benefits of UVM-SystemC methodology

- Less design time for testbench components
  - Base library provides analysis ports and callbacks
- Low learning curve for new users to the IP
  - Testbench framework well known in verification circles
- Less time in test coding for IP validation at SoC level using UVM-SV
  - Language specific updates between SC and SV via simple script
- Reduced coding time for testbench components for IP at SoC level
  - Re-use of custom bus functional model written at IP level
- Reduced man power required
  - Same owner can work on IP and SoC validation

© Accellera Systems Initiative

Sample Conversion Capabilities of the UVM-SC to UVM-SV script

- Changing class extension syntax
  - `class ahb_transaction : public uvm_randomized_sequence_item` to `class ahb_transaction extends public uvm_randomized_sequence_item`
- Updating the component phase arguments
  - `void run_phase(uvm::uvm_phase& phase)` to `function void run_phase(uvm::uvm_phase phase)`
- Modifying the constructor calls
  -
```
ahb_driver( uvm::uvm_component_name name = "ahb_driver"):
uvm::uvm_driver<ahb_transaction>( name ),ahb_pipeline_lock(1)
{ ...  }
```
    to
```
function new (string name = "ahb_driver"):
super.new(name);
endfunction
```
- Replacing loop constructor brackets with begin-end
  - `if(!uvm_config_db<ahb_if*>::get(this, "*", "vif", ahb_vif)) { …      }` to `if(!uvm_config_db<ahb_if*>::get(this, "*", "vif", ahb_vif)) begin …      end`

accellera SYSTEMS INITIATIVE

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Summary

- What went well
  - Availability of all uvm component base classes enabled fast bring up of the UVM-SystemC framework (reporting, objection handling etc.)
  - Visibility of source code helped in component development
- What could be improved
  - SCV randomization limitations with fixed point data types
  - Multiple vendor simulator support for UVM-SystemC compile/elab
  - More examples of complete validation framework will be useful for beginners (maybe put up our example for reference)

accellera SYSTEMS INITIATIVE
© Accellera Systems Initiative

Conclusion

- UVM-SystemC based validation framework enables development of **configurable**, **re-usable** and **structured** components
- standard implementation technique enables resilient testbench across multiple users
- methodology should be adopted across companies and EDA vendors to make validation truly language agnostic and enhance the UVM-SystemC VIP portfolio

© Accellera Systems Initiative

References

- UVM-SystemC
  - http://www.accellera.org/images/downloads/drafts-review/
- CRAVE
  - http://www.systemc-verification.org/crave/

Questions