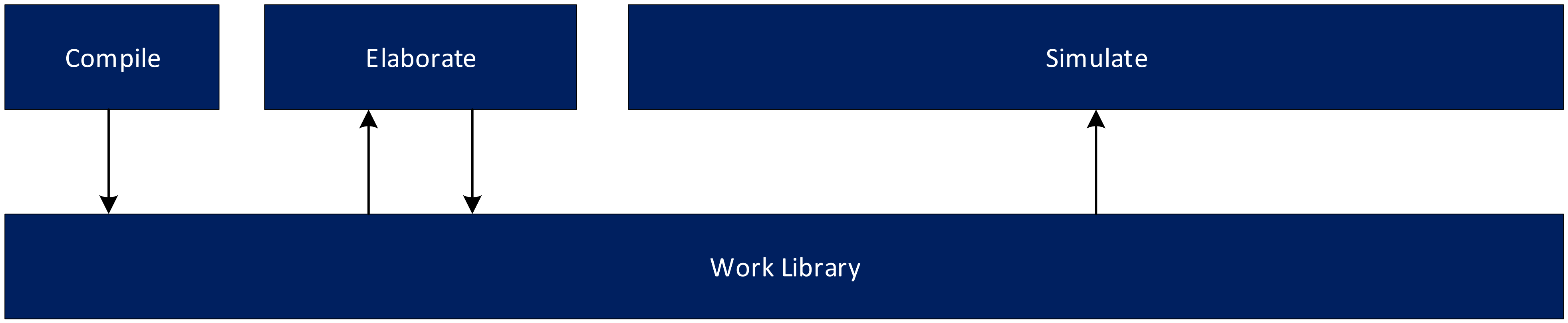
2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

It's Not Too Late to Adopt: The Full Power of UVM

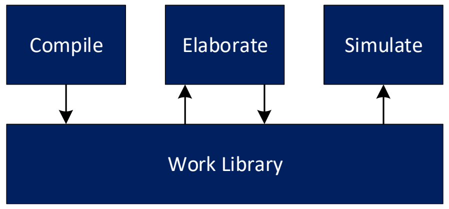
Kathleen Wittmann
Rockwell Automation

accellera SYSTEMS INITIATIVE

Simulation Flow



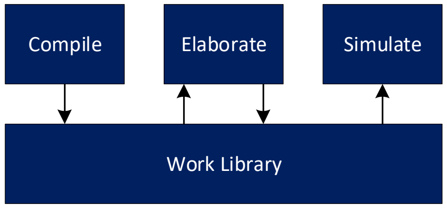
Methods for Improving Simulation Efficiency

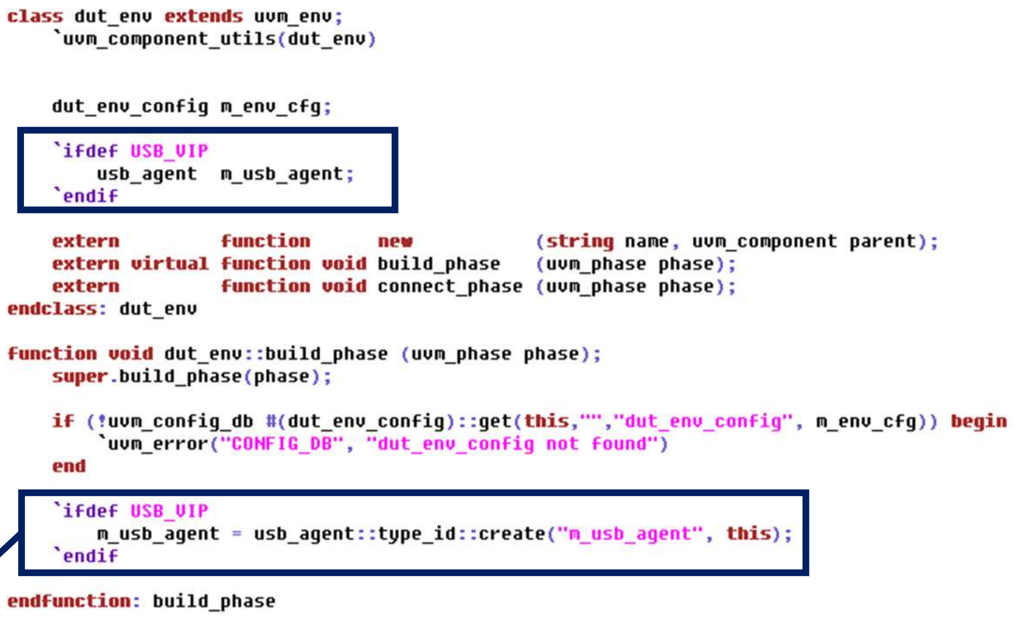


- **Incremental Compilation**:
  - Once compiled, only modified files are recompiled.
  - **Saves compilation time.**
- **Elaboration Snapshots:**
  - Design and testbench are elaborated once. Subsequent testcase runs go straight to simulation.
  - **Saves compilation and elaboration time.**
- **Simulation Snapshots:**
  - A snapshot is taken at some point in a simulation. Subsequent testcase runs begin at that point.
  - **Saves compilation, elaboration, and simulation time.**

Requirements for Incremental Compilation and Elaboration Snapshots

- Same Work Library across tests
- The same static testbench across tests

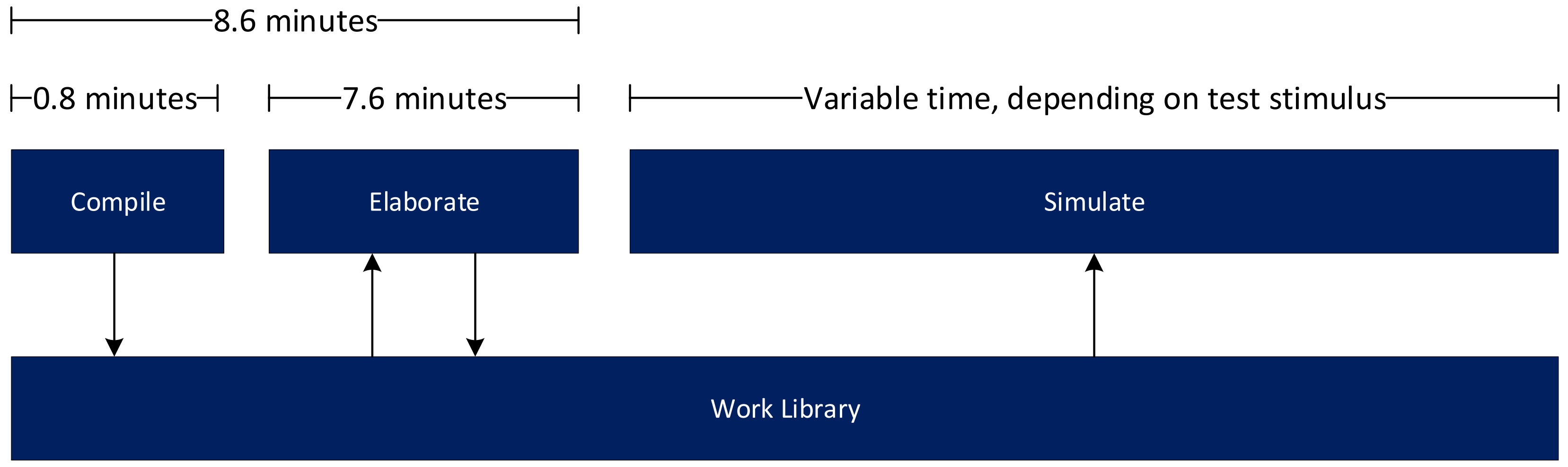




```
class dut_env extends uvm_env;
    `uvm_component_utils(dut_env)


    dut_env_config m_env_cfg;

    `ifdef USB_VIP
        usb_agent  m_usb_agent;
    `endif

    extern          function      new          (string name, uvm_component parent);
    extern virtual  function void build_phase   (uvm_phase phase);
    extern          function void connect_phase (uvm_phase phase);
endclass: dut_env

function void dut_env::build_phase (uvm_phase phase);
    super.build_phase(phase);

    if (!uvm_config_db #(dut_env_config)::get(this,"","dut_env_config", m_env_cfg)) begin
        `uvm_error("CONFIG_DB", "dut_env_config not found")
    end

    `ifdef USB_VIP
        m_usb_agent = usb_agent::type_id::create("m_usb_agent", this);
    `endif

endfunction: build_phase
```

A different static environment results, based on whether USB_VIP is defined or not

Simulation Flow



If the testbench is the same from test to test, the flow can be optimized to skip right to simulation.

In a 2000-test regression, this optimization saves nearly 12 days of processing time.

Rewritten UVM Environment

Statically Configured

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


    dut_env_config m_env_cfg;

    `ifdef USB_VIP
        usb_agent  m_usb_agent;
    `endif

    extern          function      new           (string name, uvm_component parent);
    extern virtual  function void build_phase   (uvm_phase phase);
    extern          function void connect_phase (uvm_phase phase);
endclass: dut_env

function void dut_env::build_phase (uvm_phase phase);
    super.build_phase(phase);

    if (!uvm_config_db #(dut_env_config)::get(this,"","dut_env_config", m_env_cfg)) begin
        `uvm_error("CONFIG_DB", "dut_env_config not found")
    end

    `ifdef USB_VIP
        m_usb_agent = usb_agent::type_id::create("m_usb_agent", this);
    `endif

endfunction: build_phase
```

Dynamically Configured

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

    dut_env_config m_env_cfg;

    usb_agent      m_usb_agent;

    extern          function      new           (string name, uvm_component parent);
    extern virtual  function void build_phase   (uvm_phase phase);
    extern          function void connect_phase (uvm_phase phase);
endclass: dut_env

function void dut_env::build_phase (uvm_phase phase);
    super.build_phase(phase);

    if (!uvm_config_db #(dut_env_config)::get(this,"","dut_env_config", m_env_cfg)) begin
        `uvm_error("CONFIG_DB", "dut_env_config not found")
    end

    if (m_env_cfg.has_agent_usb) begin
        uvm_config_db #(usb_agent_config)::set( this,
                                                "m_usb_agent",
                                                "usb_agent_config",
                                                 m_env_cfg.m_usb_agent_cfg)
        m_usb_agent = usb_agent::type_id::create("m_usb_agent", this);
    end

endfunction: build_phase
```

The UVM Test Configures the UVM Environment

```
class dut_env_config extends uvm_object;

  `uvm_object_param_utils(dut_env_config)

  // Agent Configuration Objects
  axi_agent_config_t       m_axi_agent_cfg;
  usb_agent_config         m_usb_agent_cfg;

  // Other Variables
  bit                      has_agent_axi   = 1;
  bit                      has_agent_usb   = 1;

  bit                      has_scoreboard  = 1;
  bit                      has_coverage    = 0;

  // Register Block
  dut_block_map            dut_rm;

  ...

  extern function new(string name = "dut_env_config");

endclass: dut_env_config
```

```
class dut_test_base extends uvm_test;
    ...
    extern         function void build_phase      (uvm_phase phase);
    extern virtual function void configure_dut_env(dut_env_config cfg);
endclass: dut_test_base

function void dut_test_base::build_phase(uvm_phase phase);
    // Create the Environment Configuration Object
    m_dut_env_cfg    = dut_env_config::type_id::create("m_dut_env_cfg");
    configure_dut_env(m_dut_env_cfg);

    // Create and configure the Agent Configuration Objects
    // Assign the Agent Configuration Objects in the Environment Configuration Object
    ...

    // Put the environment configuration in the uvm_config_db
    uvm_config_db #(dut_env_config)::set(this, "*", "dut_env_config", m_dut_env_cfg);

    // Create the environment
    m_env = dut_env::type_id::create("m_env", this);

endfunction: build_phase

function void dut_test_base::configure_dut_env (dut_env_config cfg);
    cfg.has_agent_usb = 1;
endfunction: configure_dut_env
```

UVM Test Inheritance

```
class dut_test_base extends uvm_test;
    ...
    extern         function void build_phase      (uvm_phase phase);
    extern virtual function void configure_dut_env(dut_env_config cfg);
endclass: dut_test_base

function void dut_test_base::build_phase(uvm_phase phase);
    // Create the Environment Configuration Object
    m_dut_env_cfg    = dut_env_config::type_id::create("m_dut_env_cfg");
    configure_dut_env(m_dut_env_cfg);

    // Create and configure the Agent Configuration Objects
    // Assign the Agent Configuration Objects in the Environment Configuration Object
    ...

    // Put the environment configuration in the uvm_config_db
    uvm_config_db #(dut_env_config)::set(this, "*", "dut_env_config", m_dut_env_cfg);

    // Create the environment
    m_env = dut_env::type_id::create("m_env", this);

endfunction: build_phase

function void dut_test_base::configure_dut_env (dut_env_config cfg);
    cfg.has_agent_usb = 1;
endfunction: configure_dut_env
```

<u>has_agent_usb will be 1</u>

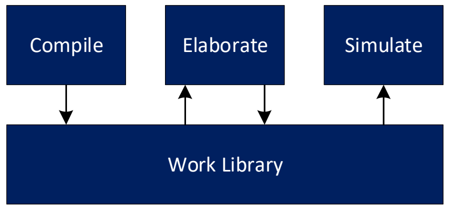
```
class test_with_usb extends dut_test_base;

    `uvm_component_utils(test_with_usb)
    ...
    extern function void build_phase (uvm_phase phase);
endclass: test_with_usb

function void test_with_usb::build_phase(uvm_phase phase);
  super.build_phase(phase);
endfunction: build_phase
```

<u>has_agent_usb will be 0</u>

```
class test_without_usb extends dut_test_base;

    `uvm_component_utils(test_without_usb)
    ...
endclass: test_without_usb

function void test_without_usb::configure_dut_env (dut_env_config cfg);
    cfg.has_agent_usb = 0;
endfunction: configure_dut_env

function void test_without_usb::build_phase(uvm_phase phase);
  super.build_phase(phase);
endfunction: build_phase
```

Requirements for Incremental Compilation and Elaboration Snapshots

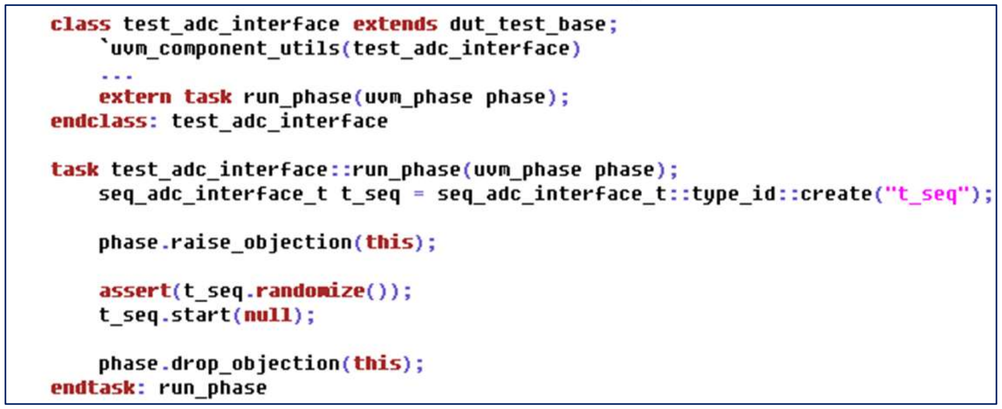
- Same Work Library across tests
- The same static testbench across tests



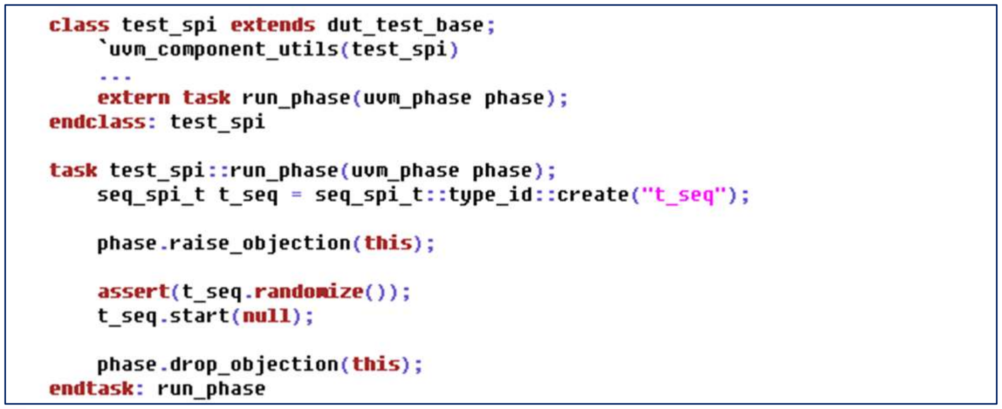
Dynamic Configuration of the UVM Environment enables these requirements to be met.

Incremental Compilation & Elaboration Snapshots are now a reality!

Simulation Snapshots through Sequence Override

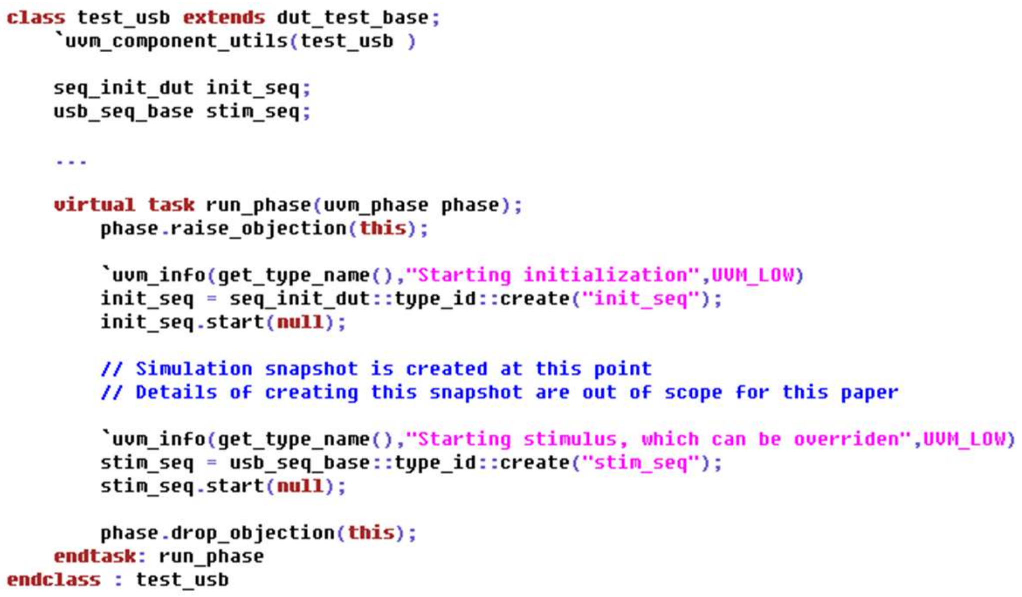


```
class test_adc_interface extends dut_test_base;
    `uvm_component_utils(test_adc_interface)
    ...
    extern task run_phase(uvm_phase phase);
endclass: test_adc_interface

task test_adc_interface::run_phase(uvm_phase phase);
    seq_adc_interface_t t_seq = seq_adc_interface_t::type_id::create("t_seq");

    phase.raise_objection(this);

    assert(t_seq.randomize());
    t_seq.start(null);

    phase.drop_objection(this);
endtask: run_phase
```



```
class test_spi extends dut_test_base;
    `uvm_component_utils(test_spi)
    ...
    extern task run_phase(uvm_phase phase);
endclass: test_spi

task test_spi::run_phase(uvm_phase phase);
    seq_spi_t t_seq = seq_spi_t::type_id::create("t_seq");

    phase.raise_objection(this);

    assert(t_seq.randomize());
    t_seq.start(null);

    phase.drop_objection(this);
endtask: run_phase
```

Instead of the 1:1 UVM Test to UVM Sequence ratio
<u>on the left, sequence override enables a 1:N ratio</u>



```
class test_usb extends dut_test_base;
    `uvm_component_utils(test_usb )

    seq_init_dut init_seq;
    usb_seq_base stim_seq;

    ...

    virtual task run_phase(uvm_phase phase);
        phase.raise_objection(this);

        `uvm_info(get_type_name(),"Starting initialization",UVM_LOW)
        init_seq = seq_init_dut::type_id::create("init_seq");
        init_seq.start(null);

        // Simulation snapshot is created at this point
        // Details of creating this snapshot are out of scope for this paper

        `uvm_info(get_type_name(),"Starting stimulus, which can be overriden",UVM_LOW)
        stim_seq = usb_seq_base::type_id::create("stim_seq");
        stim_seq.start(null);

        phase.drop_objection(this);
    endtask: run_phase
endclass : test_usb
```

Sequence Override through Inherited Sequence Classes

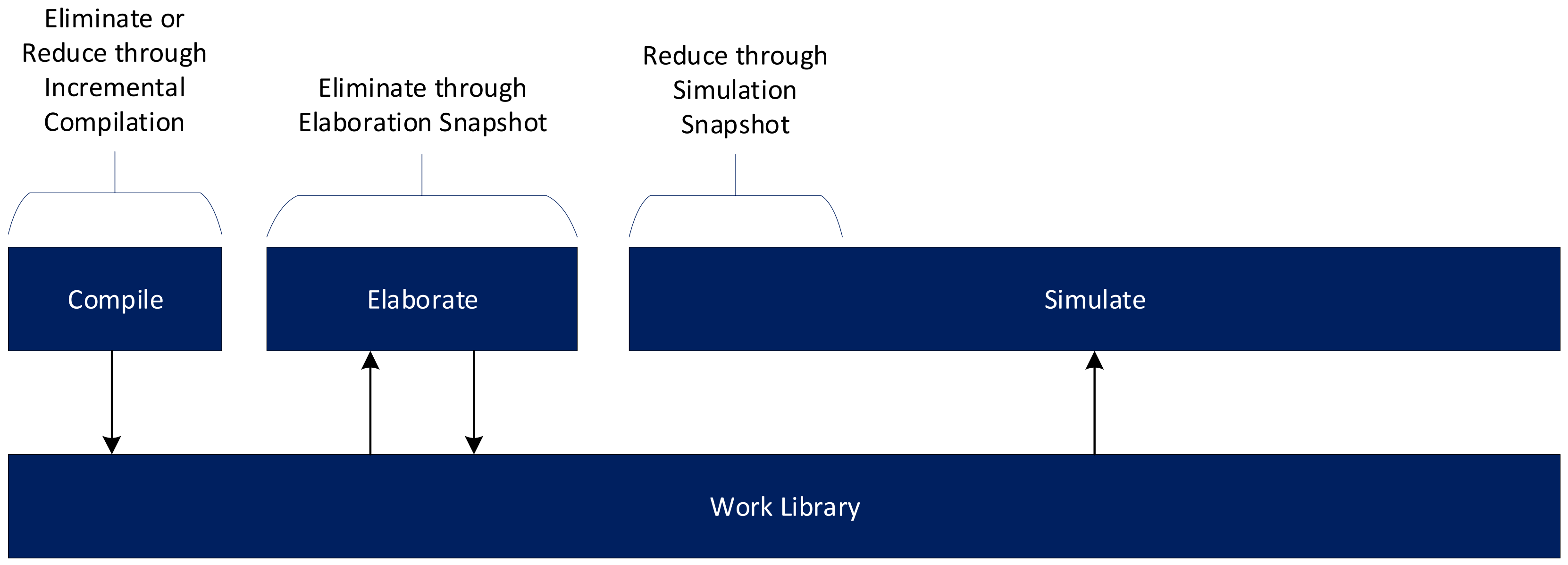
```
class test_usb extends dut_test_base;
    `uvm_component_utils(test_usb )

    seq_init_dut init_seq;
    usb_seq_base stim_seq;

    ...

    virtual task run_phase(uvm_phase phase);
        phase.raise_objection(this);

        `uvm_info(get_type_name(),"Starting initialization",UVM_LOW)
        init_seq = seq_init_dut::type_id::create("init_seq");
        init_seq.start(null);

        // Simulation snapshot is created at this point
        // Details of creating this snapshot are out of scope for this paper

        `uvm_info(get_type_name(),"Starting stimulus, which can be overriden",UVM_LOW)
        stim_seq = usb_seq_base::type_id::create("stim_seq");
        stim_seq.start(null);

        phase.drop_objection(this);
    endtask: run_phase
endclass : test_usb
```

```
class usb_seq_base extends dut_seq_base;
    `uvm_object_utils(usb_seq_base)

    // Declare USB-SPECIFIC sequencers, sequence items, sequences, and
    // register models that can be used by derived classes

    // Declare USB-SPECIFIC commonly used functions and tasks

    task body();
        // Do USB-SPECIFIC stimulus and/or create USB-SPECIFIC transactions
        // for use by derived classes
    endtask: body
endclass: usb_seq_base
```

```
class seq_exercise_dma extends usb_seq_base;
    `uvm_object_utils(seq_exercise_dma)
    ....

    task body();
        // Do some stimulus
    endtask: body
endclass: seq_exercise_dma
```

```
class seq_suspend_resume extends usb_seq_base;
    `uvm_object_utils(seq_suspend_resume)
    ....

    task body();
        // Do some stimulus
    endtask: body
endclass: seq_suspend_resume
```

Improved Simulation Efficiency


Eliminate or Reduce through Incremental Compilation
Eliminate through Elaboration Snapshot
Reduce through Simulation Snapshot
Compile
Elaborate
Simulate
Work Library

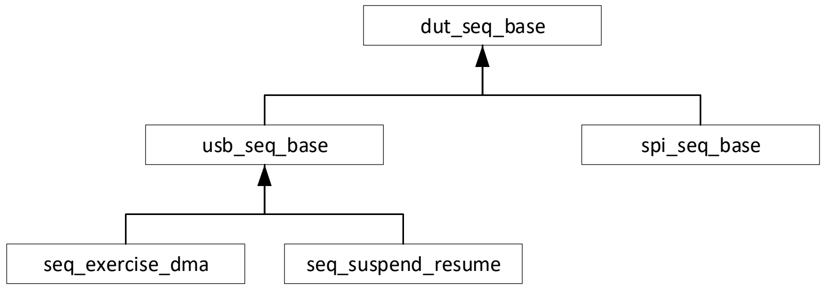
Embracing the full power of UVM can lead to simulation efficiency

Institutionalizing the Power of UVM

1. Benchmark and share the data to motivate change
2. Automate the creation of UVM testbenches
3. Document a methodology
4. Periodic knowledge share and review of the testbench

Document a Methodology: Sequence Inheritance

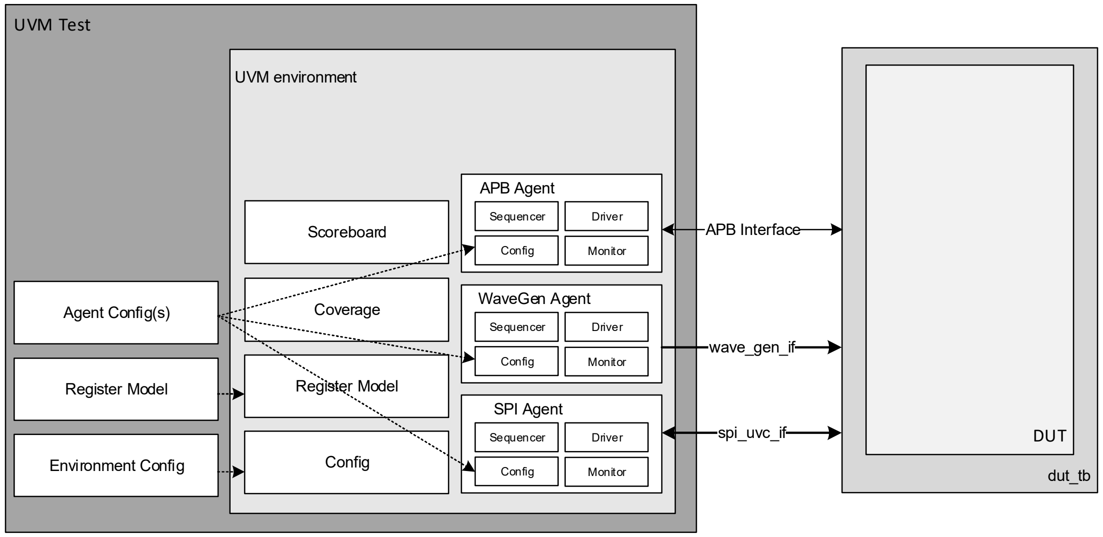
```
class dut_seq_base extends uvm_sequence #(uvm_sequence_item);
    `uvm_object_utils(dut_seq_base)

    dut_env_config m_env_cfg;

    // Declare sequencers, sequence items, sequences, and
    // register models that can be used by derived classes

    // Declare commonly used functions and tasks

    task body();
        // Get the environment configuration object
        if(!uvm_config_db #(dut_env_config)::get(null,
                                                  {"uvm_test_top.",get_full_name()},
                                                  "dut_env_config",
                                                  m_env_cfg)) begin
            `uvm_fatal("body", "dut_env_config configuration object not found")
        end

        // Create transactions
        spi_txn = spi_frame::type_id::create("spi_txn");
        ...

        // Prepare the register model for use by the derived test sequences
        dut_rm  = m_env_cfg.dut_rm;

        // Connect sequencer handles for each active agent in environment
        if (m_env_cfg.m_dut_reset_agent_cfg.is_active == UVM_ACTIVE) begin
            dut_reset_sqr = m_env_cfg.m_dut_reset_sqr;
        end
        m_spi_sqr = new[m_env_cfg.num_agent_spi];
        for (int i=0; i<m_env_cfg.num_agent_spi; i++) begin
            m_spi_sqr[i] = m_env_cfg.m_spi_sqr[i];
        end
    endtask: body
endclass: dut_seq_base
```



```
class usb_seq_base extends dut_seq_base;
    `uvm_object_utils(usb_seq_base)

    // Declare USB-SPECIFIC sequencers, sequence items, sequences, and
    // register models that can be used by derived classes

    // Declare USB-SPECIFIC commonly used functions and tasks

    task body();
        // Do USB-SPECIFIC stimulus and/or create USB-SPECIFIC transactions
        // for use by derived classes
    endtask: body
endclass: usb_seq_base
```

Document a Methodology: UVM Testbench Generation

```
module dut_tb;

    import uvm_pkg::*;
    import dut_test_pkg::*;

    // Virtual interfacce for each agent
    apb_uvc_if    vif_apb     (clk, rst_l);
    spi_uvc_if    vif_spi     (clk, rst_l);

    wave_gen_if   dut_reset_if(.clock(clk), .resetn(rst_l));

    reset_gen#(.TIME(200000), .POLARITY(0))
             i_reset_gen(.reset(rst_l));
    clk_gen  #(.PERIOD(12500), .POLARITY(0))
             i_clk_gen(.clk(clk));

    dut my_dut(
        .clk   (clk),
        .rst_l (dut_rst_l)
    );

    assign dut_rst_l = dut_reset_if.InputPin & rst_l;

    // Place the virtual interfaces in the uvm_config_db & execute run_test()
    initial begin
        uvm_config_db #(virtual wave_gen_if)::set(null, "uvm_test_top", "vif_dut_reset", dut_reset_if);
        uvm_config_db #(virtual apb_uvc_if)::set( null, "uvm_test_top", "vif_apb",       vif_apb);
        uvm_config_db #(virtual spi_uvc_if)::set( null, "uvm_test_top", "vif_spi",       vif_spi);

        run_test();
    end
endmodule: dut_tb
```

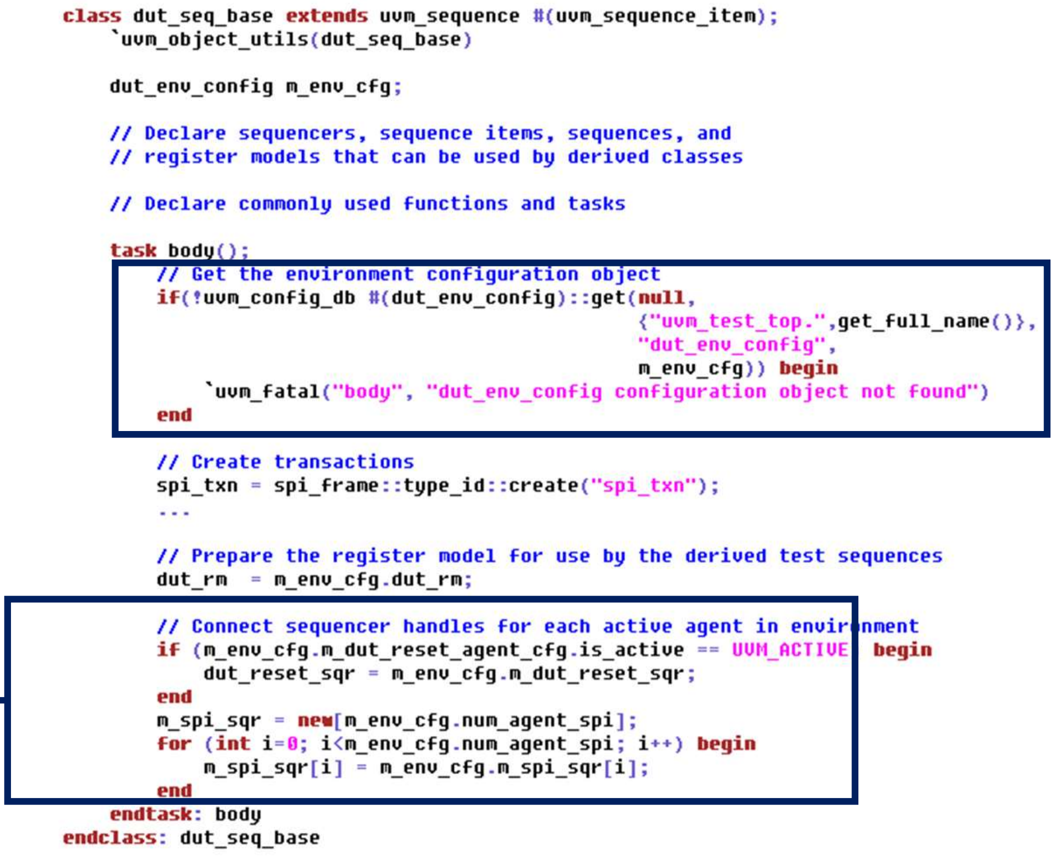


UVM Test Base Class

1) Create the agent configuration
2) Get its virtual interface out of the uvm_config_db
3) Assign the virtual interface to the agent configuration
4) Further configure the agent

```
class dut_test_base extends uvm_test;
    dut_env                m_env;
    dut_env_config         m_dut_env_cfg;

    wave_gen_agent_config m_dut_reset_agent_cfg;
    apb_agent_config      m_apb_agent_cfg;
    spi_agent_config      m_spi_agent_cfg;
    ...
    extern         function void build_phase      (uvm_phase phase);
    extern virtual function void configure_dut_env(dut_env_config cfg);
endclass: dut_test_base

function void dut_test_base::build_phase(uvm_phase phase);
    // Create the Environment Configuration Object
    m_dut_env_cfg    = dut_env_config::type_id::create("m_dut_env_cfg");
    configure_dut_env(m_dut_env_cfg);

    // Create and configure the Agent Configuration Objects
    // Assign the Agent Configuration Objects in the Environment Configuration Object
    m_spi_agent_cfg = spi_agent_config::type_id::create("m_spi_agent_cfg");
    if(!uvm_config_db #(virtual spi_uvc_if)::get(this, "", "vif_spi", m_spi_agent_cfg.vif)) begin
        `uvm_error("RESOURCE_ERROR", "spi_uvc_if virtual interface not found")
    end
    m_dut_env_cfg.m_spi_agent_cfg = m_spi_agent_cfg;
    configure_spi_agent(m_spi_agent_cfg);
    ...

    // Put the environment configuration in the uvm_config_db
    uvm_config_db #(dut_env_config)::set(this, "*", "dut_env_config", m_dut_env_cfg);

    // Create the environment
    m_env = dut_env::type_id::create("m_env", this);

endfunction: build_phase

function void dut_test_base::connect_phase(uvm_phase phase);
    // Connect sequencer handles for each active agent
    if (m_dut_env_cfg.m_spi_agent_cfg.is_active == UVM_ACTIVE) begin
        m_dut_env_cfg.m_spi_sqr = m_env.m_spi_agent.m_sequencer;
    end
  ...
endfunction: connect_phase
```

Importance of the Configuration Object to Sequences



```
class dut_seq_base extends uvm_sequence #(uvm_sequence_item);
    `uvm_object_utils(dut_seq_base)

    dut_env_config m_env_cfg;

    // Declare sequencers, sequence items, sequences, and
    // register models that can be used by derived classes

    // Declare commonly used functions and tasks

    task body();
        // Get the environment configuration object
        if(!uvm_config_db #(dut_env_config)::get(null,
                                                 {"uvm_test_top.",get_full_name()},
                                                 "dut_env_config",
                                                 m_env_cfg)) begin
            `uvm_fatal("body", "dut_env_config configuration object not found")
        end

        // Create transactions
        spi_txn = spi_frame::type_id::create("spi_txn");
        ...

        // Prepare the register model for use by the derived test sequences
        dut_rm  = m_env_cfg.dut_rm;

        // Connect sequencer handles for each active agent in environment
        if (m_env_cfg.m_dut_reset_agent_cfg.is_active == UVM_ACTIVE) begin
            dut_reset_sqr = m_env_cfg.m_dut_reset_sqr;
        end
        m_spi_sqr = new[m_env_cfg.num_agent_spi];
        for (int i=0; i<m_env_cfg.num_agent_spi; i++) begin
            m_spi_sqr[i] = m_env_cfg.m_spi_sqr[i];
        end
    endtask: body
endclass: dut_seq_base
```

Sequence retrieves sequencers from the environment configuration object

Importance of the Configuration Object to the Environment

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

    dut_env_config m_env_cfg;

    usb_agent      m_usb_agent;

    extern         function      new          (string name, uvm_component parent);
    extern virtual function void build_phase   (uvm_phase phase);
    extern         function void connect_phase (uvm_phase phase);
endclass: dut_env

function void dut_env::build_phase (uvm_phase phase);
    super.build_phase(phase);

    if (!uvm_config_db #(dut_env_config)::get(this,"","dut_env_config", m_env_cfg)) begin
        `uvm_error("CONFIG_DB", "dut_env_config not found")
    end

    if (m_env_cfg.has_agent_usb) begin
        uvm_config_db #(usb_agent_config)::set( this,
                                                "m_usb_agent",
                                                "usb_agent_config",
                                                 m_env_cfg.m_usb_agent_cfg)
        m_usb_agent = usb_agent::type_id::create("m_usb_agent", this);
    end

endfunction: build_phase
```

Environment builds components specified in the environment configuration object

Summary

- Some macros do have value (e.g. ASIC vs. FPGA)
- Use UVM to dynamically configure your environment
- The **virtual** keyword is powerful!
- The configuration object plays a critical part in configuring a UVM Environment for a particular UVM Test
- Sequence override adds even more power to UVM
- Take advantage of methods that improve simulation efficiency
- Realizing "The Full Power of UVM" requires an investment in knowledge, methodology, and infrastructure

Questions?