Agenda: Day 2

DAY 2

5 UVM Configuration & Factory

6 UVM Component Communication 

7 UVM Scoreboard & Coverage

8 UVM Callback 

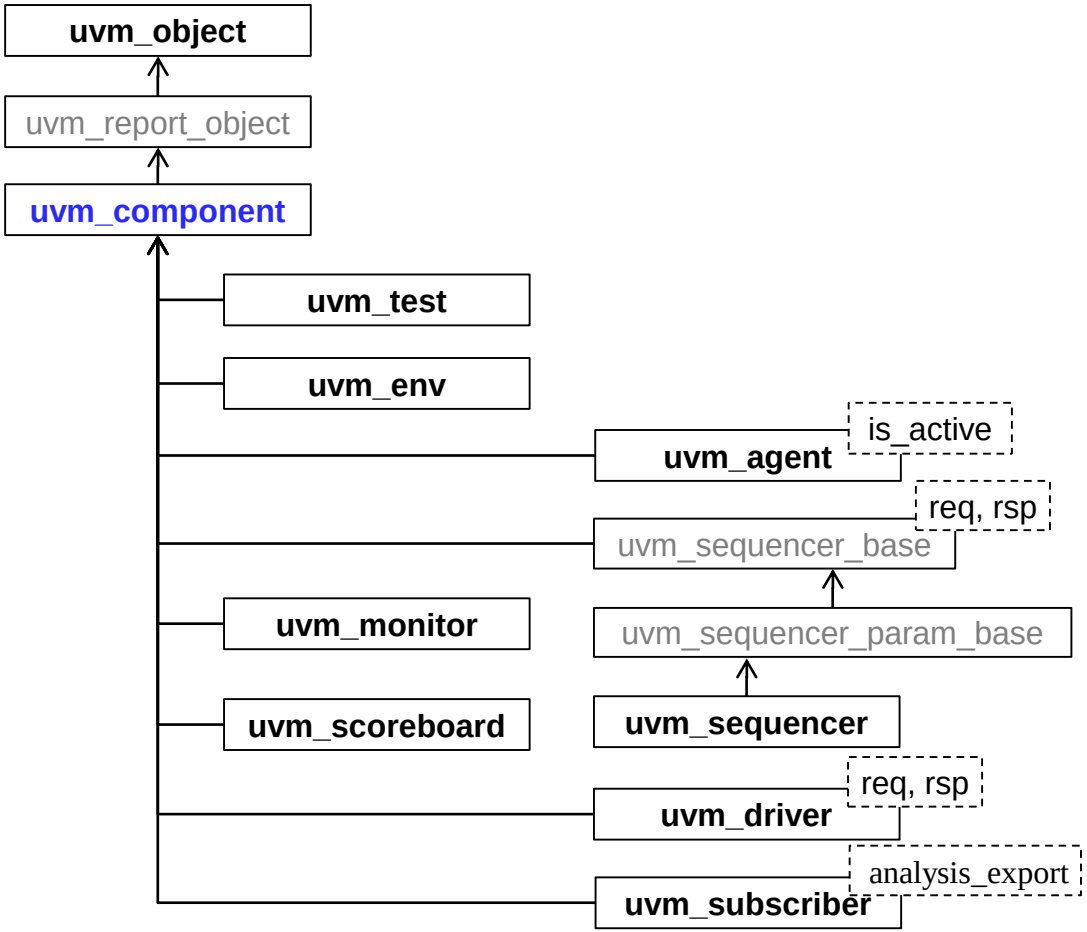
Unit Objectives

After completing this unit, you should be able to:

- Describe component logical hierarchy
- Use logical hierarchy to get/set component configuration fields
- Use factory to create test replaceable transaction and components

UVM Component Base Class Structure

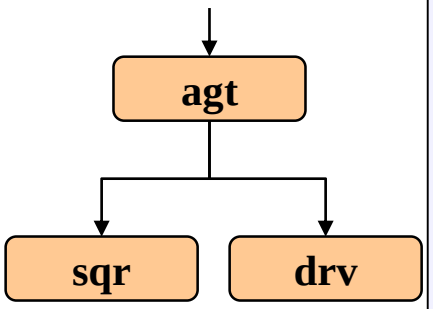
- **Behavioral base class is `uvm_component`**
  - Has logical parent-child relationship
    - Used for phasing control, configuration and factory override




```
build
connect
end_of_elaboration
start_of_simulation
run*
extract
check
report
final
```

Component Parent-Child Relationships

- **Logical relationship is established at creation of component object**




```
class driver extends uvm_driver #(packet); // utils macro
  function new(string name, uvm_component parent);
    super.new(name, parent);
  endfunction
  ...;
endclass

class agent extends uvm_agent; // utils macro
  typedef uvm_sequencer #(packet) sequencer;
  sequencer sqr; driver drv;
  function new(string name, uvm_component parent); ...;
  function void build_phase(…); super.build_phase(…);
    sqr = sequencer::type_id::create("sqr", this);
    drv = driver::type_id::create("drv", this);
  endfunction
endclass
...
agent agt = agent::type_id::create("agt", this);
```

Pass parent in via constructor (components only)

Establish logical hierarchy

Display & Querying

- **Rich set of methods for query & display**

Display properties

```
agt.print();
```

Get logical name

```
string str = drv.get_name();
```

Get hierarchical name

```
string str = drv.get_full_name();
```

Find one component via logical name

```
uvm_component comp;
comp = uvm_root::get().find("*.sqr");
```

Logical name

Can use wildcard
* – match anything (including ".")
+ – one or more character (including ".")
? – match one character

Find all matching components

```
uvm_component comps[$];
uvm_root::get().find_all("*.drv?", comps);
foreach (comps[i]) begin
  comps[i].print();
end
```

Query Hierarchy Relationship

- **Easy to get handle to object parent/children**

Get handle to parent

```
uvm_component comp;
comp = this.get_parent();
```

Finding object via logical name

Logical name

```
uvm_component comp;
comp = vip.get_child("sqr");
```

Determine number of children

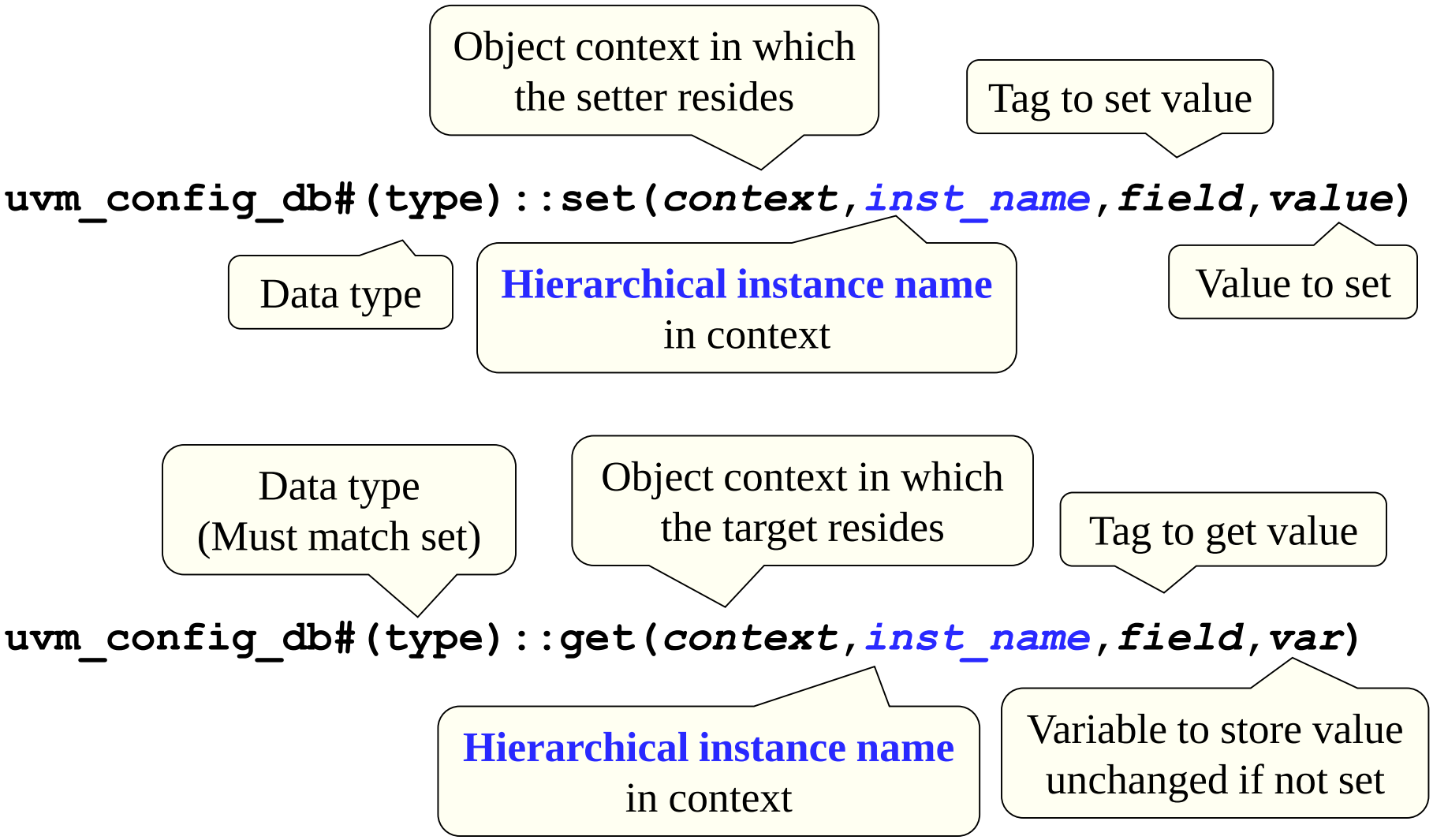
```
int num_ch = vip.get_num_children();
```

Iterate through children

```
string name;
uvm_component child;
if (vip.get_first_child(name)) do begin
  child = vip.get_child(name);
  child.print();
end while (vip.get_next_child(name));
```

Use Logical Hierarchy in Configuration

- **Mechanism for configuring object properties**



Component Configuration Example

- **Agent component field configuration**
  - Should target agent
    - Agent then configures child components

Configure agents in **build_phase** of environment

```
class router_env extends uvm_env;
  // utils macro and constructor not shown
  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    uvm_config_db #(int)::set(this, "i_agt[10]", "port_id", 10);
  endfunction
```

Populate value in configuration database targeting agent

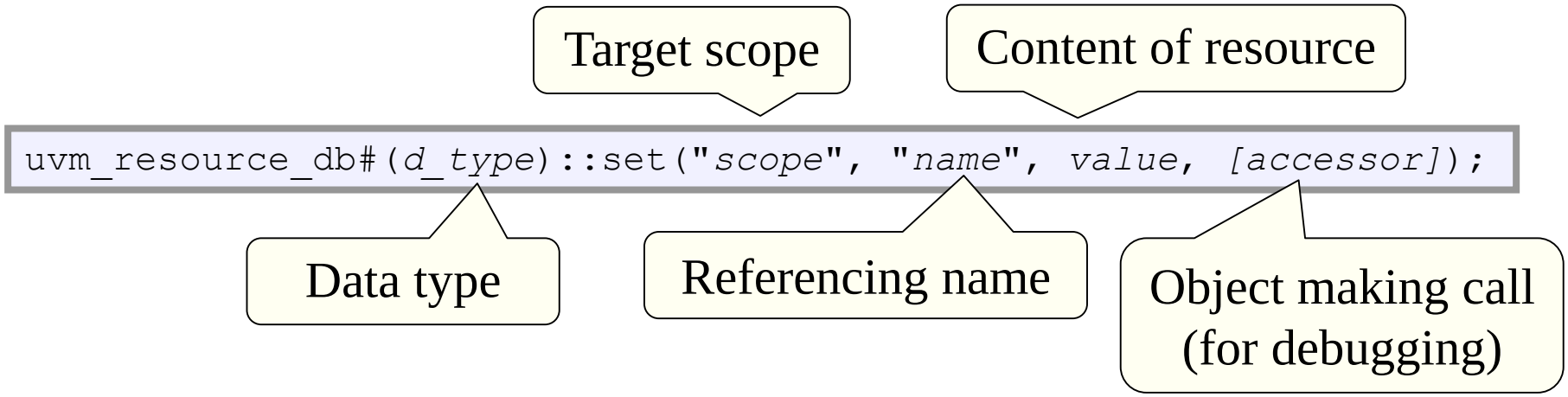
```
class agent extends uvm_agent;
  // constructor not shown
  int port_id = -1;   // default value
  `uvm_component_utils_begin(agent)
    `uvm_field_int(port_id, UVM_ALL_ON)
  `uvm_component_utils_end
  virtual function void build_phase(…); ...
    uvm_config_db #(int)::get(this, "", "port_id", port_id);
    uvm_config_db #(int)::set(this, "*", "port_id", port_id);
  endfunction
endclass
```

Retrieve configuration in agent

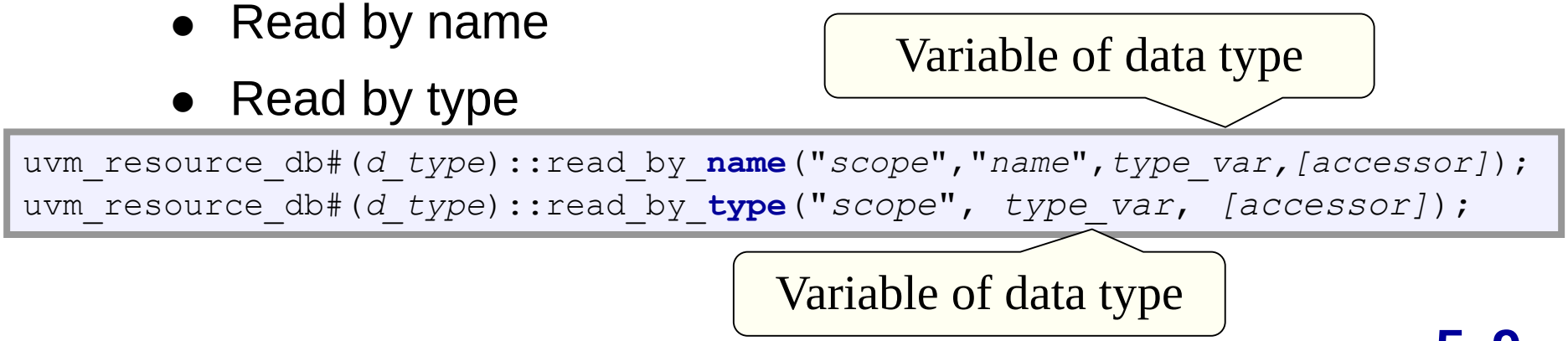
Then, set child component configuration

UVM Resource

- **`uvm_config_db` targets instances of objects**
- **For global configuration, use `uvm_resource_db`**



- **Retrieval of the resources can be done in two ways**
  - Read by name
  - Read by type



Manage DUT Interface Configuration

- **In program/module, use `uvm_resource_db::set()`**

```
program automatic test;
import uvm_pkg::*;

initial begin
  uvm_resource_db#(virtual router_io)::set("router_vif", "",
                                           router_test_top.router_if);
  uvm_resource_db#(virtual reset_io)::set("reset_vif", "",
                                          router_test_top.reset_if);

  $timeformat(-9, 1, "ns", 10);
  run_test();
end
endprogram
```

In program, access interface via XMR
In module, access interface directly

`uvm_resource_db` is used instead of **`uvm_config_db`** to isolate the program/module code from test hierarchy changes. If test hierarchy changes, the program/module code is not impacted.

Test Configures Agents with Interfaces

- In test, use `uvm_resource_db::read_by_type()` to retrieve interface
- Use `uvm_config_db::set()` to configure targeted agents

```
class test_base extends uvm_test; // other code left off
  virtual router_io router_vif;
  virtual reset_io  reset_vif;
  virtual function void build_phase(uvm_phase phase); // other code left off
    uvm_resource_db#(virtual reset_io)::read_by_type("reset_vif",
                                                     reset_vif, this);
    uvm_config_db#(virtual reset_io)::set(this, "env.r_agt",
                                          "vif", reset_vif);
    uvm_resource_db#(virtual router_io)::read_by_type("router_vif",
                                                     router_vif, this);
    uvm_config_db#(virtual router_io)::set(this, "env.i_agt[*]",
                                          "vif", router_vif);
  endfunction
endclass
```

Configure agent, NOT children of agent!

Configuring Component's Interface (1/2)

- **In driver/monitor**
  - Call **`uvm_conig_db#()::get()`** in **`build_phase`**
  - Check for correctness in **`end_of_elaboration_phase`**

```
class driver extends uvm_driver#(packet); // other code not shown
  virtual router_io vif;

  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    uvm_config_db#(virtual router_io)::get(this,"","vif",vif))
  endfunction

  virtual function void end_of_elaboration_phase(uvm_phase phase);
    super.end_of_elaboration_phase(phase);
    if (vif == null) begin
      `uvm_fatal("CFGERR", "Driver DUT interface not set");
    end
  endfunction
endclass
```

Configuring Component's Interface (2/2)

- **In agent**
  - In **`build_phase`**
    - Call **`uvm_conig_db#()::get()`** to retrieve interface
    - Call **`uvm_conig_db#()::set()`** to set interface for children of agent

```
class input_agent extends uvm_agent;   // other code not shown
  virtual router_io vif;
  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    uvm_config_db#(virtual router_io)::get(this, "", "vif", vif);
    uvm_config_db#(virtual router_io)::set(this, "*", "vif", vif);
  endfunction
endclass
```

Additional Needs: Manage Test Variations

- **Tests need to introduce class variations, e.g.**
  - Adding constraints
  - Modify the way data is sent by the driver
- **Variation can be instance based or global**
- **Control object construction for all or specific instances of a class**
- **Create generic functionality**
  - Deferring exact object creation to runtime

Solution : Built-in UVM Factory

Test Requirements: Transaction

- **How to manufacture *transaction* instances with additional information without modifying the original source file?**

```
class monitor extends uvm_monitor;
  ...;
  virtual task run_phase(uvm_phase phase);
    forever begin
      packet pkt;
      pkt = new("pkt");
      get_packet(pkt);
    end
  endtask
endclass
```

Type of object determines memory allocated for the instance

Poor coding style
No way of overriding the transaction object with a derived type

Impossible to add new members or modify constraint later

```
class bad_packet extends packet; ...;
  bit bad;
  virtual function int compute_crc();
endclass
```

Test Requirements: Components

- **How to manufacture *component* instances with additional information without modifying the original source file?**

```
class input_agent extends uvm_agent;
  packet_sequencer sqr;
  driver drv;
  ...
  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    sqr = new("sqr", this);
    drv = new("drv", this);
    ...
  endfunction
endclass
```

No way of modifying the component behavior

```
class NewDriver extends driver;
  virtual task run_phase(uvm_phase phase);
    if (dut.vif.done == 1)
      ...
  endtask
endclass
```

Impossible to change behavior for test

Factories in UVM

Implementation flow

- **Factory instrumentation/registration**
  - `uvm_object_utils(*Type*)
  - `uvm_component_utils(*Type*)

Macro creates a proxy class (called **`type_id`**) to represent the object/component **and** registers an instance of the proxy class in uvm_factory

- **Construct object using static proxy class method**
  - *ClassName obj* = *ClassName*::type_id::create(…);

Use proxy class to create object

- **Class overrides**
  - set_type_override_by_type(…);
  - set_inst_override_by_type(…);

Proxy class can create objects specified in test with overrides

Transaction Factory

- **Construct object via `create()` using factory class**

Required! Macro defines a proxy class called **`type_id`**
An instance of proxy class is registered in **`uvm_factory`**

```
class packet extends uvm_sequence_it
  rand bit[3:0] sa, da;
  `uvm_object_utils_begin(packet)
    `uvm_field_int(sa, UVM_ALL_ON)
  ...;
endclass
```

```
class monitor extends uvm_monitor;
  task run_phase(uvm_phase phase);
    forever begin
      packet pkt;
      pkt = packet::type_id::create("pkt", this);
      ...;
    end
  endtask
endclass
```

Component handle optional

Use proxy's **`create()`** method
to construct transaction object

UVM Factory Transaction Creation

```
class packet extends uvm_sequence_item;
  rand bit[3:0] sa, da;
  `uvm_object_utils_begin(packet)
    ...
endclass
```

packet.sv

macro expansion

```
class packet extends uvm_sequence_item;
  typedef uvm_object_registry #(packet, "packet") type_id;
  ...
endclass
```

proxy to represent packet class

Parent component path establishes search path

```
packet pkt;
pkt = packet::type_id::create("pkt", this);
```

Any overrides of class **packet**?

Y

```
bad_packet ord = new("pkt");
pkt = ord;
```

N

```
pkt = new("pkt");
```

Component Factory

- **Construct object via `create()` using factory class**
  - Similar to transaction creation

```
class driver extends uvm_driver #(packet);
  `uvm_component_utils(driver)
  ...
endclass
```

Required! Macro defines a proxy class called **`type_id`**
An instance of proxy class is registered in uvm_factory

```
class router_env extends uvm_env;
  `uvm_component_utils(router_env)
  driver drv;
  ...
  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    drv = driver::type_id::create("drv", this);
    ...
  endfunction
endclass
```

Parent component handle **required**!

Use **`create()`** method of proxy class to construct component object

UVM Factory Component Creation

```
class driver extends uvm_driver #(packet);
  `uvm_component_utils(driver)
  ...
endclass
```

driver.sv

```
class driver extends uvm_driver #(packet);
  typedef uvm_component_registry #(driver, "driver") type_id;
  ...
endclass
```

macro expansion

```
driver drv;
drv = driver::type_id::create("drv", this);
```

Any overrides of class **driver** under “this” parent scope?

Y

```
newDrv ord;
ord = new("drv", this);
drv = ord;
```

N

```
drv = new("drv", this);
```

Command-line Override

- **User can override factory objects at command line**
  - Objects must be constructed with the factory ***class_name*`::type_id::create(...)`** method

`+uvm_set_inst_override=<`***req_type***`>,<`***override_type***`>,<`***inst_path***`>`

`+uvm_set_type_override=<`***req_type***`>,<`***override_type***`>`

- **Works like the overrides in the factory**
  - **`set_inst_override()`**
  - **`set_type_override()`**
- **Example:**

`+uvm_set_type_override=`***driver,newDriver***

`+uvm_set_inst_override=`***packet,my_pkt,****`.agt.*`

No space character!

Override in Test

- **User can chose type of override**
  - **`set_inst_override_by_type()`**
  - **`set_type_override_by_type()`**

```
class test_override extends test_base;
  // utils and constructor not shown
  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    set_type_override_by_type(driver::get_type(),
                                          newDriver::get_type());
    set_inst_override_by_type("*.agt.sqr.*", packet::get_type(),
                                             my_pkt::get_type());


  endfunction
endclass
```

Only packet instances in matching sequencers are now packet_da_3

Unit Objectives Review

Having completed this unit, you should be able to:

- **Describe component logical hierarchy**
- **Use logical hierarchy to get/set component configuration fields**
- **Use factory to create test replaceable transaction and components**

Appendix

- Accessing DUT signals via DPI
- Sequence Configuration
- Variable Length Configuration
- Enum Configuration
- Configuration Debugging
- Disabling Auto Field Configuration

Accessing DUT Signals via DPI

Direct DUT Signal Access

- **From UVM uvm_hdl.svh file**

```
// For all functions, returns 1 if the call succeeded, 0 otherwise.

// uvm_hdl_deposit: sets the given HDL ~path~ to the specified ~value~.
function int uvm_hdl_deposit(string path, uvm_hdl_data_t value);

// uvm_hdl_read: gets the value at the given ~path~.
function int uvm_hdl_read(string path, output uvm_hdl_data_t value);

// uvm_hdl_force: forces the ~value~ on the given ~path~.
function int uvm_hdl_force(string path, uvm_hdl_data_t value);

// uvm_hdl_force_time: forces the ~value~ on the given ~path~ for the specified amount of
// ~force_time~. If ~force_time~ is 0, <uvm_hdl_deposit> is called.
task uvm_hdl_force_time(string path, uvm_hdl_data_t value, time force_time=0);

// uvm_hdl_release_and_read: releases a value previously set with <uvm_hdl_force>.
// ~value~ is set to the HDL value after the release.
function int uvm_hdl_release_and_read(string path,inout uvm_hdl_data_t value);

// uvm_hdl_release: releases a value previously set with <uvm_hdl_force>.
function int uvm_hdl_release(string path);
```

Sequence Configuration

Configuring Sequences (Instance-Based)

- **Set in test**

```
class test_20_items extends test_base; // other code not shown
  virtual function void build_phase(…); super.build_phase(…);
    uvm_config_db#(int)::set(this, "env.i_agt.sqr.packet_sequence",
                             "item_count", 20);
  endfunction
endclass
```

Field tag within sequence

Value

Name path to sequence (may not be class name)

- **Reference configuration field through get_full_name()**

```
class packet_sequence extends sequence_base; // other code not shown
  int item_count = 10;
  function new(string name = "packet_sequence");
  virtual task pre_start(); super.pre_start();
    uvm_config_db#(int)::get(null, this.get_full_name(),
                             "item_count", item_count);
  endtask

  virtual task body();
    repeat(item_count) begin ...; end
  endtask
endclass
```

Full path to sequence

Field tag to retrieve value

Variable to store value unchanged if not set

Configuring Sequences (Class-Based)

- **Set in test**

```
class test_20_items extends test_base; // other code not shown
  virtual function void build_phase(…); super.build_phase(…);
    uvm_config_db#(int)::set(this, "env.i_agt.sqr.packet_sequence",
                             "item_count", 20);
  endfunction
endclass
```

Name path to class type

- **Reference configuration field through get_type_name()**

```
class packet_sequence extends sequence_base; // other code not shown
  int item_count = 10;
  function new(string name = "packet_sequence");
  virtual task pre_start(); super.pre_start();
    uvm_config_db#(int)::get(get_sequencer(), this.get_type_name(),
                             "item_count", item_count);
  endtask

  virtual task body();
    repeat(item_count) begin ...; end
  endtask
endclass
```

Class type

Sequencer path

Recommended

Configuring Sequences (Sequencer-Based)

- **Set in test**

```
class test_20_items extends test_base;
  // utils and constructor not shown
  virtual function void build_phase(…); super.build_phase(…);
    uvm_config_db#(int)::set(this, "env.i_agt.sqr",
                             "item_count", 20);
  endfunction
endclass
```

Sequencer configuration field

Name path to sequencer

- **Reference configuration field through get_sequencer()**

```
class packet_sequence extends sequence_base; // macros not shown
  int item_count = 10;
  function new(string name = "packet_sequence"); // code not shown

  virtual task body();
    uvm_config_db#(int)::get(this.get_sequencer(), "",
                             "item_count", item_count);
    repeat(item_count) begin ...; end
  endtask
endclass
```

Sequencer path

Sequencer configuration field

Configuring Sequences (Agent-Based)

- **Provides access to agent configuration for sequences**
  - Only to be used to retrieve agent configuration
- **Set in test**

```
class test_agent_configuration extends test_base;
  // utils and constructor not shown
  virtual function void buid_phase(…); ...
    uvm_config_db#(int)::set(this, "env.i_agt", "port_id", 3);
  endfunction
endclass
```

Agent configuration field

Name path to agent

- **Retrieve agent configuration field through sequencer**

```
class packet_sequence extends sequence_base; // other code not shown
 int port_id;
 virtual task body();
  uvm_sequencer_base my_sqr = get_sequencer();
  uvm_config_db#(int)::get(my_sqr.get_parent(), "", "port_id",port_id);
  ...
 endtask
endclass
```

Full path to agent

Agent configuration field

Configuration Debugging

Debugging Configuration Issues (1/4)

- **Common issues**
  - Configuration is strongly typed
  - Mismatched types between the set and get calls will not flag an error during get

```
uvm_config_db#(int)::set::(this,"env","item_count",10);
uvm_config_db#(int unsigned)::get(this,"","item_count", item_count);
```

  - Mismatched field names and/or scopes due to typos and change in hierarchy will not flag an error

Debugging Configuration Issues (2/4)

- **Always check return status**
  - When doing a **`read_by_name/_type()`** or **`get()`** check the return status and print an error message if configuration is required

```
if(!uvm_config_db#(int unsigned)::get(this,"","item_count", item_count))
  `uvm_info("CFGERR", $sformatf("item_count not available for %s",
                                this.get_full_name()), UVM_MEDIUM);
```

  - Use run-time switch available to trace all config sets and gets

```
+UVM_CONFIG_DB_TRACE
+UVM_RESOURCE_DB_TRACE
```

  - Use tracing controls in test code
    - `uvm_config_db_options::turn_on_tracing() // turn tracing on`
    - `uvm_config_db_options::turn_off_tracing() //turn tracing off`
    - `uvm_config_db_options::is_tracing() //check status of tracing`

Debugging Configuration Issues (3/4)

- **Global handle available to UVM resource pool**

  `uvm_resources`

- **Dumping resource data**
  - Dump all resources in the resource pool using **`dump()`**

```
uvm_resources.dump(.audit(1));  //audit==1 displays resource access details

Sample output:
default_sequence [/^uvm_test_top\.env\.i_agt\.sqr\.main_phase$/] : (class
    uvm_pkg::uvm_object_wrapper) ?
-
  --------
  uvm_test_top.env.i_agt reads: 0 @ 0.0ns  writes: 1 @ 0.0ns
  uvm_test_top.env.i_agt.sqr reads: 1 @ 0.0ns  writes: 0 @ 0.0ns
```

  - Dump all 'gets' done using **`uvm_resource_db::read_by_name()/_type()`**

```
uvm_resources.dump_get_records();
```

Debugging Configuration Issues (4/4)

- **Find all unused resources (on which no gets were performed)**
  - See code in notes section

```
uvm_resources.find_unused_resources();
```

Sample Output of code below:

```
===Unused Resources===

delay [/^uvm_test_top\.env$/] : (int) 5
---------------------------------------------------------------------------------------------------
Name                    Type              Size  Value
---------------------------------------------------------------------------------------------------
delay                   <unknown>           -     @510
---------------------------------------------------------------------------------------------------

  uvm_test_top.env reads: 0 @ 0.0ns  writes: 1 @ 0.0ns
```

UVM Resource and Config DB Debug

- Within your code you can enable dumping
  - resources.dump(audit);
- Command line based debug
  - ./simv +UVM_RESOURCE_DB_TRACE +UVM_CONFIG_DB_TRACE

```
UVM_INFO /fs/Release/linux_RH4_AMD64_TD_32_debug_Engineer/etc/uvm-1.1/base/uvm_resource_db.svh(130) @
    0.0ns: reporter [CFGDB/GET] Configuration 'uvm_test_top.recording_detail' (type logic
    signed[4095:0]) read by uvm_test_top = null (failed lookup)
UVM_INFO /fs/Release/linux_RH4_AMD64_TD_32_debug_Engineer/etc/uvm-1.1/base/uvm_resource_db.svh(130) @
    0.0ns: reporter [CFGDB/GET] Configuration 'uvm_test_top.recording_detail' (type int) read by
    uvm_test_top = null (failed lookup)
UVM_INFO @ 0.0ns: reporter [RNTST] Running test test_seq_lib_cfg...
UVM_INFO @ 0.0ns: reporter [UVM_CMDLINE_PROC] Applying config setting from the command line:
    +uvm_set_config_int=uvm_test_top.env,foo,3
UVM_INFO /fs/Release/linux_RH4_AMD64_TD_32_debug_Engineer/etc/uvm-1.1/base/uvm_resource_db.svh(130) @
    0.0ns: reporter [CFGDB/SET] Configuration 'uvm_test_top.env.foo' (type logic signed[4095:0]) set
    by  = (logic signed[4095:0]) 11
UVM_INFO /fs/Release/linux_RH4_AMD64_TD_32_debug_Engineer/etc/uvm-1.1/base/uvm_resource_db.svh(130) @
    0.0ns: reporter [CFGDB/GET] Configuration 'uvm_test_top.env.recording_detail' (type logic
    signed[4095:0]) read by uvm_test_top.env = null (failed lookup)
UVM_INFO /fs/Release/linux_RH4_AMD64_TD_32_debug_Engineer/etc/uvm-1.1/base/uvm_resource_db.svh(130) @
    0.0ns: reporter [CFGDB/GET] Configuration 'uvm_test_top.env.recording_detail' (type int) read by
    uvm_test_top.env = null (failed lookup)
```