Easy uvm_config_db use

A simplified and reusable uvm_config_db methodology for environment developers and test writers

Bob Oden, UVM Field Specialist, Mentor Graphics, bob_oden@mentor.com

Motivation

- Provide a mechanism for sharing resources within a simulation that provides features needed by architects and simplicity needed by test writers.
- Architects need a mechanism that can allow resource access by hierarchy and general scope.
- Test writers need access to simulation resources while treating environment as a black box.

The Resource Table

- Contains all information required by test writers to access environment resources.
- Independent of environment implementation. The environment is a black box to test writers.
- Creates a clear association between an interface, its virtual interface handle, and the sequencer used to place traffic on the interface.
- Control waveform viewing of transactions on interfaces in the table without recompilation

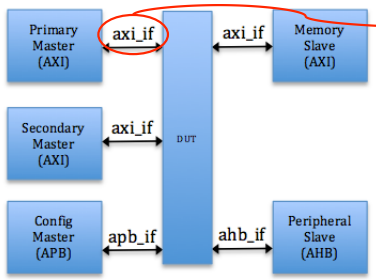


Figure 1: Example Design

Interface Description	Type	Transaction	Identifier
Primary AXI Master	axi_if	axi_transaction	primary_master
Secondary AXI Master	axi_if	axi_transaction	secondary_master
Config APB Master	apb_if	apb_transaction	config_master
AXI Memory Slave	axi_if	axi_transaction	memory_slave
PeripheralAHB Slave	ahb_if	ahb_transaction	peripheral_slave

Table 1: Example Design Resource Table

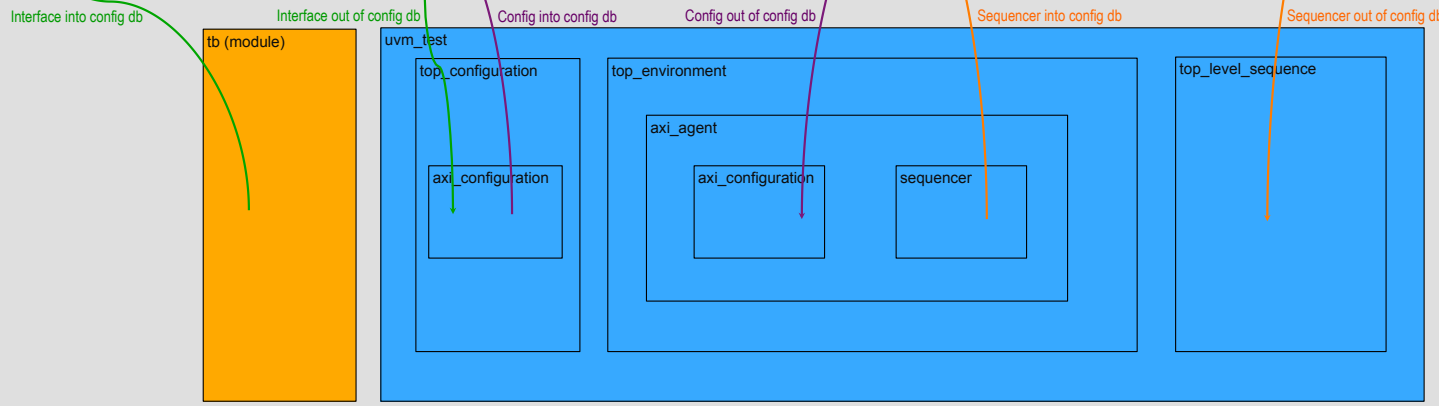
```
// In a test level parameters package
// This code defines the unique string used to identify each interface in the resource database.
// A parameter is used to eliminate runtime errors due to typing errors.
//
parameter string PRIMARY_MASTER   = "primary_master";
parameter string SECONDARY_MASTER = "secondary_master";
parameter string CONFIG_MASTER    = "config_master";
parameter string MEMORY_SLAVE     = "memory_slave";
parameter string PERIPHERAL_SLAVE = "peripheral_slave";
```

```
// In the module where the interfaces reside
uvm_config_db #( virtual axi_if )::set(
                    null , "VIRTUAL_INTERFACES" , PRIMARY_MASTER,   primary_axi_master_bus );
uvm_config_db #( virtual axi_if )::set(
                    null , "VIRTUAL_INTERFACES" , SECONDARY_MASTER, secondary_axi_master_bus );
uvm_config_db #( virtual apb_if )::set(
                    null , "VIRTUAL_INTERFACES" , CONFIG_MASTER,    config_apb_master_bus );
uvm_config_db #( virtual axi_if )::set(
                    null , "VIRTUAL_INTERFACES" , MEMORY_SLAVE,     axi_memory_slave_bus );
uvm_config_db #( virtual ahb_if )::set(
                    null , "VIRTUAL_INTERFACES" , PERIPHERAL_SLAVE, peripheral_ahb_slave_bus );
```

```
// In the agents configuration class
// Each protocol would have its own configuration class: axi_configuration, apb_configuration, etc.
class axi_configuration extends uvm_object;
   // This string variable holds this agents interface identifier
   string interface_name;
   // This variable holds the virtual interface handle
   virtual axi_if axi_bus_vif;
   // This function is used to configure the agent
   function void configure_interface(string interface_name, string path_to_agent)
      // Store the interface identifier string for use when placing sequencer in uvm_config_db
      this.interface_name = interface_name;
      // Get this agents interface from the uvm_config_db
      uvm_config_db #( virtual axi_if )::get(
                          null, "VIRTUAL_INTERFACES", interface_name, axi_bus_vif );
      // Make this configuration available to the agent through the uvm_config_db
      uvm_config_db #( axi_configuration )::set( null ,path_to_agent, "AGENT_CONFIG", this);
```

```
// In the agents class declaration file
// Each protocol would have its own agent class: axi_agent, apb_agent, etc.
class axi_agent extends uvm_agent #(…
   // This variable is a handle to this agents configuration class
   axi_configuration axi_config;
   function void build_phase (uvm_phase phase);
      // Get this agents configuration
      uvm_config_db #( axi_configuration )::get( this , "" , "AGENT_CONFIG" ,  axi_config );
      …
      // Place this agents sequencer in the uvm_config_db using the interface identifier
      uvm_config_db #( uvm_sequencer #(axi_transaction) )::set(
                          null , "SEQUENCERS" , axi_config.interface_name, axi_sequencer );
```

```
      // In the top level sequence
// Define and instantiate sequencer handles
typedef uvm_sequencer #(axi_transaction) axi_sequencer_t;
axi_sequencer_t primary_master_sequencer, secondary_master_sequencer, memory_slave_sequencer;
typedef uvm_sequencer #(apb_transaction) apb_sequencer_t;
apb_sequencer_t config_master_sequencer;
typedef uvm_sequencer #(ahb_transaction) ahb_sequencer_t;
ahb_sequencer_t peripheral_slave_sequencer;
// Retrieve each sequencer from the uvm_config_db
uvm_config_db #( axi_sequencer_t )::get(
                    null, "SEQUENCERS", PRIMARY_MASTER,   primary_master_sequencer );
uvm_config_db #( axi_sequencer_t )::get(
                    null, "SEQUENCERS", SECONDARY_MASTER, secondary_master_sequencer );
uvm_config_db #( apb_sequencer_t )::get(
                    null, "SEQUENCERS", CONFIG_MASTER,    config_master_sequencer );
uvm_config_db #( axi_sequencer_t )::get(
                    null, "SEQUENCERS", MEMORY_SLAVE,     memory_slave_sequencer );
uvm_config_db #( ahb_sequencer_t )::get(
                    null, "SEQUENCERS", PERIPHERAL_SLAVE, peripheral_slave_sequencer );
```



Results

- Architects have needed features
  - Pass resources by hierarchy or generic scope
  - Supports component and environment reuse
  - Supports simulation and emulation
  - Use uvm_resource_db if desired
- Test writers have needed simplicity
  - All necessary information is in Resource Table
  - Environment is a black box
  - No time invested learning environment implementation
  - Focus on writing tests