**Module: UVM-2**

**Register Abstraction Layer**

**RAL**

Assignment 04

* **Write( ) method:**

| virtual task write(  output uvm\_status\_e status,  input uvm\_reg\_data\_t value,  input uvm\_path\_e path = UVM\_DEFAULT\_PATH,  input uvm\_reg\_map map = null,  input uvm\_sequence\_base parent = null,  input int prior = -1,  input uvm\_object extension = null,  input string fname = "",  input int lineno = 0); |
| --- |

The **write()** method in UVM is used to write a specific value to a register or memory in the DUT through the UVM Register Abstraction Layer (RAL). It can perform both front-door access (via an address map) and back-door access (directly to DUT).

Here is a detailed explanation of the purpose of each argument in the write() method:

### **Arguments in the write() Method**

1. **output uvm\_status\_e status:** This is an output variable that returns the status of the write() operation.
2. **input uvm\_reg\_data\_t value:** This is the value that you want to write to the register or memory.
3. **input uvm\_path\_e path = UVM\_DEFAULT\_PATH:** Specifies the access path to be used for the write operation.
4. **input uvm\_reg\_map map = null:** Specifies the address map to be used for the write operation.
5. **input uvm\_sequence\_base parent = null:** Identifies the parent sequence initiating the write operation.
6. **input int prior = -1:** Specifies the priority of the write operation. If set to -1 (default), the default priority for the sequence is used.
7. **input uvm\_object extension = null:** Allows passing additional user-defined data during the write operation.
8. **input string fname = "":** Indicates the filename where the write() call is made.
9. **input int lineno = 0:** Indicates the line number in the file where the write() call is made.

* **Register Write Sequences:**

| **cntrl\_wr\_sequence** |
| --- |
| class cntrl\_wr\_sequence extends uvm\_sequence;    // Factory Registration  `uvm\_object\_utils(cntrl\_wr\_sequence)  // Register Model  reg\_blk reg\_model;  // Constructor  function new( string name = "cntrl\_wr\_sequence");  super.new(name);  endfunction : new  // Body Task  task body();  uvm\_status\_e status;  bit [31:0] wdata;  wdata = $urandom();  reg\_model.ctrl.write(status, wdata);  endtask    endclass : cntrl\_wr\_sequence |

| **reg1\_wr\_sequence** |
| --- |
| class reg1\_wr\_sequence extends uvm\_sequence;    // Factory Registration  `uvm\_object\_utils(reg1\_wr\_sequence)  // Register Model  reg\_blk reg\_model;  // Constructor  function new( string name = "reg1\_wr\_sequence");  super.new(name);  endfunction : new  // Body Task  task body();  uvm\_status\_e status;  bit [31:0] wdata;  wdata = $urandom();  reg\_model.r1.write(status, wdata);  endtask    endclass : reg1\_wr\_sequence |

| **reg2\_wr\_sequence** |
| --- |
| class reg2\_wr\_sequence extends uvm\_sequence;    // Factory Registration  `uvm\_object\_utils(reg2\_wr\_sequence)  // Register Model  reg\_blk reg\_model;  // Constructor  function new( string name = "reg2\_wr\_sequence");  super.new(name);  endfunction : new  // Body Task  task body();  uvm\_status\_e status;  bit [31:0] wdata;  wdata = $urandom();  reg\_model.r2.write(status, wdata);  endtask    endclass : reg2\_wr\_sequence |

| **reg3\_wr\_sequence** |
| --- |
| class reg3\_wr\_sequence extends uvm\_sequence;    // Factory Registration  `uvm\_object\_utils(reg3\_wr\_sequence)  // Register Model  reg\_blk reg\_model;  // Constructor  function new( string name = "reg3\_wr\_sequence");  super.new(name);  endfunction : new  // Body Task  task body();  uvm\_status\_e status;  bit [31:0] wdata;  wdata = $urandom();  reg\_model.r3.write(status, wdata);  endtask    endclass : reg3\_wr\_sequence |

| **reg4\_wr\_sequence** |
| --- |
| class reg4\_wr\_sequence extends uvm\_sequence;    // Factory Registration  `uvm\_object\_utils(reg4\_wr\_sequence)  // Register Model  reg\_blk reg\_model;  // Constructor  function new( string name = "reg4\_wr\_sequence");  super.new(name);  endfunction : new  // Body Task  task body();  uvm\_status\_e status;  bit [31:0] wdata;  wdata = $urandom();  reg\_model.r4.write(status, wdata);  endtask    endclass : reg4\_wr\_sequence |