References	Title
IEEE Std 1800.2™-2017	Standard for Universal Verification Methodology Language Reference Manual
IEEE Std 1800™-2012	Standard for SystemVerilog Unified Hardware Design, Specification, and Verification Language

Note: By its nature, this document cannot be complete. Please refer to the above standards for full details.

Verilog Operators:

```
+     addition
-     subtraction
*     multiplication
/     division
**    exponent
%     modulus
>     greater than
<     less than
>=    grater than or equal
<=    less than or equal
==    logical equality
!=    logical inequality
===   4-logic value equality
!==   4-logic value inequality
&&    logical and
||    logical or
!     logical negation
&     bit-wise and « a & b  »
&     bit-wize unary and reduction «  &a   »
~&    unary nand reduction
|     bit-wise or « a | b »
|     bit-wize unary or reduction « |a »
~|    reduction nor
^     bit-wise exclusive or « a^b »
^     bit-wize unary or reduction « ^a »
~^    bit-wise equivalence (^~) (also unary reduction
xnor)
~     bit-wise complement
>>    bit-wise logical right shift
<<    bit-wise logical left shift
>>>   bit-wise arithmetic right shift
<<<   bit-wise arithmetic left shift
? :   condition ? value-if-true : value-if-false
*     compare equal on group of bits
=     assignment
<=    non blocking assignment (in clocked processes)
( )   grouping parenthesis, module instantiation,
        function and task call
[ ]   range as in [31:0]
{ }   concatenation { a , 2'b00 , b[2:0] }
{ <n> { } }   repeated concatenation { 16 { 2'b01 } }
```

Literals:

```
Base:
   d = decimal, o = octal, b = binary , h = hexadecimal

Default is decimal: -?[0-9]+
       Examples: 0, 1973, -123

Literals with base: [<sign>][<nr_bits>]'<Base><value>
       Examples: 'h900D , 'hcafe ,
                 12'd1239 , -'d12 , 10d123
                 'b00110100 , 4'b0001
```

Verilog Types:

```
4-Value logics of Verilog are defined by: 0,1,x,z

Net Data Types:
       wire,
       supply0, supply1, tri, triand,
       trior, tri0, tri1, wand, wor
Variable Data Types:
       reg, integer, real, realtime, time
Vectors
       wire [31:0] a;
       reg [15:0] b;
Arrays
       integer a[0:100];
       wire [31:0] a [0:1000];
       reg  [31:0] a [0:1000][1:20];
```

SystemVerilog Types

```
shortint   2-state data type, 16-bit signed integer
int        2-state data type, 32-bit signed integer
longint    2-state data type, 64-bit signed integer
byte       2-state data type, 8-bit signed integer or ASCII character
bit        2-state data type, user-defined vector size, unsigned
logic      4-state data type, user-defined vector size, unsigned
reg        4-state data type, user-defined vector size, unsigned
integer    4-state data type, 32-bit signed integer
time       4-state data type, 64-bit unsigned integer
string     str.len() ,      str.putc(i,c),   str.getc(i)
           str.itoa(i),     str.hextoa(i),   str.bintoa(i)
           str.tolower(),   str.toupper(),   str.compare()
           str.substr(i,j)
enum {red, yellow, green} light;
typedef [ enum|struct|union|class ] type_identifier;
typedef enum { red, yellow, green } color_t;
```

Control Statements:

```
if(<condition>)
    begin
      <statements>
    end

if(<condition1>)
    begin
      <statements>
    end
  else
    begin
      <statements>
    end

case(<expression>)
    <expression1> : <statement>
    <expression2> : begin
                      <statements>
                    end
    <expression3>,              // note comma
    <expression4>: <statement> // for both
         ...
    default : <statement>      // optional
endcase


for ( var=<init_val> ; <end_cond> ; <var = var + incr>)
  begin
    <statements>
  end

repeat(<count>)
  begin
    <statements>
  end

foreach( my_array[idx_var] )
  begin
    <statements using my_array[idx_var]>
  end

while(<condition>)
  begin
    <statements>
  end

do
  begin
    <statements>
  end
while(<condition>);
```

Building Blocks:

```
module my_module_name(…);
    parameter <name>=<default value>;

    input <signame>;
    output <signame>;

    assign <wire_name> = <expression>;

    initial  // zero or more initial blocks
      begin
        <sequential statements>
      end

    always  // zero or more always blocks
      begin
        <sequential statements>
      end
    task do_something;
    endtask
    function [15:0] max_value;
    endfunction

    generate ;
       genvar i ;
       for ( i = 0 ; i<10; i++)
       begin : my_name
         <always, initial, instances, signals, interface>
       end
    endgenerate
endmodule
```

Delay Statements:

```
`timescale <timeunit>/<precision>;

wait(<condition>) #(<optional_delay) <statement>
wait(<condition>) // waits for condition to become true

@(<some_event>) <statement>
@(<some_event>)
  begin
    <statements>
    end
@(posedge <signal>);
@(negedge <signal>);
#20;  // delay 20 time units
```

Blocking vs Non-Blocking Assigments

Blocking : Assignment is performed before moving to next delta cycle
Non-Blocking: Assignment is delayed to the next delta cycle

```
// Non Blocking Assignement
always @(posedge clk)
    begin
      a <= b;
      b <= a;
    end
// Blocking Assignment - Not Synthesieable
always (@posedge clk)
    begin
      tmp = b;
      b = a;
      a = tmp;
    end
// Blocking Assigment - Combinatorial Logic
always @(a or b)
  begin
    if ( a == 12 )
      c = 0 ;
    else
      c = a + b + 15 ;
  end
```

Task / Functions:

```
task do_something;
   input  REQ;
   output GNT;
   begin
     // statements. Can consume time
   end
endtask

task do_something(      input int DATA,
                        ouput logic[12:0] RESULT) ;
     // statements. Can consume time
endtask

function [15:0] max_value;
      input [15:0] a, b;
      begin
        max_value = a < b ? b : a;
      end
  endfunction

  function bit[15:0] max_value(bit[15:0] ,bit[15:0]);
        return a < b ? b : a;
  endfunction
```

Agregate Data Types

Structures and Unions

```
struct { color_t pix; int a } varA;
varA.pix = yellow;
typedef struct {
        bit[7:0] opcode;
        bit[23:0] addr;
} instruction_s;
instruction_s IR;
IR.opcode = 0'hAE;
typedef union { int i; shortreal f; } num;
```

Packed and Unpacked Arrays

```
bit [7:0] c1;
real u [7:0];
int Array[0:7][0:31];
int Array[8][32];
logic [7:0] mem [0:255];    // declares a memory array
                            // of 256 8-bit elements.
                            // The array indices are 0 to 255
mema[5] = 0;                // Write to word at address 5
data = mema[addr];          // Read word in mem at index = addr
```

Operation on Arrays

Assignments

```
A = B
A[i:j] = B[i:j]
A[x+:c] = B[y+:c]
A[i] = B[i]
```

Comparison

```
A==B, A[i:j] != B[i:j]
```

Dynamic Arrays

```
bit [3:0] nibble[]; // Dynamic array of 4-bit vectors
integer mem[2][];   // Fixed-size unpacked array composed
                    // of 2 dynamic subarrays of integers
nibble = new[12];   // allocate 12 elements in nibble
// allocate 12 elements in nibble and
// initialize the first two values to 1 and 5
nibble = new[12]( '{ 1 , 5} );

// allocate 13 elements in nibble and initialize
// the first twelve values to the previous ones
nibble = new[13]( nibble );

int nr_elem = nibble.size(); // returns nr of elements
nibble.delete();    // remove all elements of the array
```

Associative Arrays

```
integer i_array[*];         // associative array of integer
                            // (unspecified index)
bit [20:0] array_b[string]; // associative array
                            // of 21-bit vector, indexed by string
event ev_array[myClass];    // associative array
                            // of event indexed by class myClass

array_b["toto"] = 21'h13cafe;
i_array.delete(12);
int varB = i_array.size();
if ( array_b.exists("titi") )
        array_b.delete("titi");
```

Others:

```
        .first(index) , .last(index) , .next(index)
```

Associative Array Methods	Description
.delete(T idx)	Remove the element indexed by idx
.size()	Returns the size of the array
.exists(T idx)	Returns 1 if the list contains an element indexed by idx
.first(T idx)	Returns the first element indexed by idx
.last(T idx)	Relturns the ast element indexed by idx
.next(idx)	Returns the next element of the array

Queues

```
byte  q1[$];                // A queue of bytes
string names[$] = { "Bob" }; // A queue of strings
                            // with one element
integer Q[$] = { 3, 2, 7 }; // An initialized queue
                            // of integers
bit q2[$:255];              // A queue whose maximum size is 256 bits
q2.push_front(1);
bit first_bit_in = q2.pop_back();
```

Queue Methods:

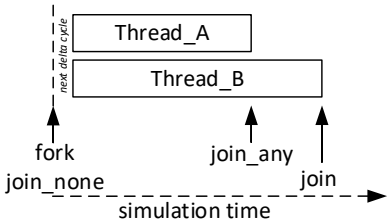
```
.size() , .insert(index, elem) , .delete(index)
.pop_front() ,      .pop_back()
.push_front(),      .push_back()
```

Pullup , PullDown

```
pullup  : default is 1 unless driven low
pullown : default is 0 unless driven high

module my_sensor( inout dinout );
        pullup (pull1) my_pullup_inst (dinout);
        logic data_enable;
        logic data_out;
        logic data_in;
        assign dinout = (data_enable == 1'b1) ?
                        data_out : 1'bz;
        assign data_in = dinout;
```

Threads

```
forever  // never ending loop
  begin
    <statements>
  end
fork
    <statement>
    <statement>
    ...
join          // wait until all statements complete
// join_none  // wait for none of the threads
// join_any   // wait for any of the thread to complete

process myjob;
fork
    begin
        myjob = process::self();
      <statements>
    end
   begin : thread_B
     <statements>
    end
join_any // wait until first statement finishes
disable thread_B; // kill the branch thread_B above
disable fork; // kill all forked sub-threads
myjob.kill(); // kill first thread of the above fork
```




```
myjob.await() ; myjob.suspend() ; myjob.status();
myjob.resume();
```

Clocking Blocks

Can be defined within modules, checkers, interfaces

```
clocking @(posedge clk);
        default input #1step output #1ns;
        input #1ps Q;          // sampled 1ps before the posedge of clk
        input #1step rst;      // sampled 1 step, the last known value
                               // ...before the clock edge
        output #1ns A;         // driven 1ns after the clock edge
        output #2 B;           // driven 1 "timeunit" after the clock edge
        output #0 sel;         // driven on the clock edge
endclocking // clk

initial
begin
    rst <= 0;
    ##1 rst <= 1;       // driven #1ns after clock
    ##1 rst <= 0;       // driven #1ns after clock
    ##2 A <= 1;         // wait for two clocks then drive #1ns
                        // after clock
```

Mailbox & Semaphores

```
semaphore sem = new(1); // initialiaze sem
                        // with 1 available key
mailbox #(string) msg = new;
string my_msg_str;

fork
   begin
      sem.get(); // get the only one key
      msg.put("Hi there !"); // send a message
      #1ns;                  // take some time
      sem.put() ;            // give back the key
   end
   begin
      sem.get(); // get the key. If not available, wait
      // key is now available,
      // check if we have a message
      if ( msg.num() >= 1 )
         my_msg_str = msg.get(); // "Hi there !"
      sem.put(); // give back the key
   end
join_any
```

Others:

```
sem.put(); sem.get(); sem.try_put(); sem.try_get()
msg.put(); msg.get(); msg.try_put(); msg.try_get();
msg.num()
```

Assertions

```
wire request , grant;
property is_granted(req , gnt);
        @(posedge clk)
        req |=> gnt;
endproperty

default clocking @(posedge clk);
endclocking
property is_granted_bis(req , gnt);
        req |=> gnt;
endproperty

ASRT_REQGNT : assert property is_granted(request ,
grant);
sequence evt_seq_A;
        req && cmd == 1 ## req == 1 |-> cmd == 2;
endsequence
sequence evt_seq_B;
        req |-> cmd == 2 ##[0:10] req == 1 && cmd == 2;
endsequence
cover sequence evt_seq_A;
```

Checkers

```
checker my_checker(event clk, logic[7:0] a, b);
  logic [7:0] sum;
  always_ff @(clk) begin
    sum <= a + 1'b1;
    p0: assert property (sum < `MAX_SUM);
  end
  p1: assert property (@clk sum < `MAX_SUM);
  p2: assert property (@clk a != b);
  p3: assert #0 ($onehot(a));
endchecker


module m(wire [31:0] bus, logic clk);
  // ...
  my_checker chk_inst0(posedge clk, data , sum);
endmodule
module regmodel_chk (...);
...
endmodule
bind regfile regmodel_chk regchk_i (.*);
```

Interfaces

```
interface apb_if(input PCLK, input PRESETn);
        logic PENABLE;
        logic PREADY;
        logic PSEL;

        modport master (
                output PENABLE ,
                output PSEL ,
                input PREADY);

        modport slave (
                input PENABLE ,
                input PSEL ,
                output PREADY);
        task drive_reset();
        //...
        endtask

        covergroup apb_if_cg @(posedge PCLK);
        // ..
        endgroup

        generate ; …
        endgenerate

        apb_if_cg cg = new();
endinterface

module dut(
        apb_if.master mif, // master modport
        apb_if.slave sif );// slave modport
        ///...
endmodule
```

Classes

```
class myBaseClass;
        int c1 = 1;
        int c2 = 1;
        int c3 = 1;
        function new(int a);
                c2 = 2;
                c3 = a;
        endfunction
        function void set_and_shift(int val);
                c3 = c2; c2 = c1;
                c1 = val;
        endfunction
endclass
class myExtendedClass extends myBaseClass;

        local int d1 = 4;
        protected int d2 = c2;

        function new;
                super.new(d3);
                this.d1 = m();
        endfunction

        external virtual function int m();

        static task T();
                ...
        endtask
endclass

function int myExtendedClass::m();
        // out of block implementation
        // of m() functions declared as external
endfunction

// Parameterized class
class E #(type T = int) extends myBaseClass;
        T my_statck[$];
        function void add( T val );
        endfunction
endclass
```

Object Instances

```
myBaseClass b;           // just a null pointer
myExtendedClass d = new; // create d object of
                         //     class MyExtendedClass
myExtendedClass::T(); // call the static task of class D
d.T();                // call the static task of object d

Class Casting:
c = d;  // implicit cast to parent class
MyExtendedClass d2;
$cast(d2,c); // cast to a subclass MyExtendedClass
```

Random Variables & Constraints

```
typedef enum {low, mid, high} AddrType;
class myBusTrans;
    rand bit[15:0] addr;
    rand bit[31:0] data;
    rand int x;
    constraint word_align {addr[1:0] == 2'b0;}
    rand AddrType atype;
    constraint addr_range
    {
        (atype == low ) -> addr inside { [0 : 15] };
        (atype == mid ) -> addr inside { [16 : 127]};
        (atype == high) -> addr inside {[128 : 255]};
    }
    // change random distribution
    // value 100 with weight = 1/3
    // value 200 with weight = 2
    // value 300 with weight = 5/5 = 1
    x dist {
        [100:102] := 1, // 1 chance out of 4 of being 100
        200       := 2, // 2 chances out of 4 of being 200
        [300:304] :/ 5  // 1 chances out of 4 of being
                        //  … in [300 to 304]
    }
    function void pre_randomize();
        // pre-randomization code
        // executed just before random values are…
        // … generated
    endfunction

    function void post_randomize();
        // post-randomization code
        // executed just after random values are generated
    endfunction
endclass
```

Random Generation

```
myBusTrans tr = new;
bit [15:0] data = 14;
tr.word_align.constraint_mode(0); // disable constraint
tr.data.rand_mode(0); // disable randomization of data

repeat (50) begin
        if ( tr.randomize() with
           { .atype == mid;
             tr.addr == 15 || tr.addr == 18;
             tr.data == local::data;
           } )
                $display ("addr = %16h data = %h\n",
                                tr.addr, tr.data);
        else
                $display ("Randomization failed.\n");
end

randcase
        3 : x = 1;  // 3/8 chance to execute x=1
        1 : x = 2;  // 1/8 chance to execute x=2
        4 : x = 3;  // 4/8 chance to execute x=3
endcase
```

```
int a, b;
randcase
        a + b    : begin x = 1; a+= 1; end
        a - b    : begin x = 2; b-=2; end
        a + 2*b  : x = 3;
        12'b800  : x = 4;
endcase

randsequence( main )
        main : first second done ;
        first : add | dec ;
        second : pop | push ;

        done : { $display("done"); } ;
        add : { $display("add"); } ;
        dec : { $display("dec"); } ;
        pop : { $display("pop"); } ;
        push : { $display("push"); } ;
endsequence
```

Functional Coverage

```
covergroup cg1 @(posedge clk); … endgroup
covergroup cg2 @my_event0; … endgroup

my_covergroup cg0 = new;

always @(posedge clock)
    if( sampling_condition )  cg0.sample();
integer val_operand;

covergroup cg;
    coverpoint val_operand {
      bins a = { [0:63] , 65 };   // single bins
      bins b[] = { 100,101,102 }; // [] array: 3 bins
    }
endgroup

bit [31:0] register_12;
covergroup uart_cg;
    parity_en : coverpoint register_12[7];
    parity    : coverpoint register_12[6:5];
endgroup

typedef enum { INIT, A, B, C } state_t;
state_t cur_state;
state_t prev_state;

covergroup cg;
    cur_state  : coverpoint cur_state;
    prev_state : coverpoint prev_state;
    cur_trans  : coverpoint cur_state {
      bins multiple[] = (INIT,A,B,C [* 3:4] );
      bins transition[] =
                (INIT,A,B,C => INIT,A,B,C );
      bins seq = default sequence ;
    }
    cross prev_state, cur_state;
endgroup
```

DPI

C-Code

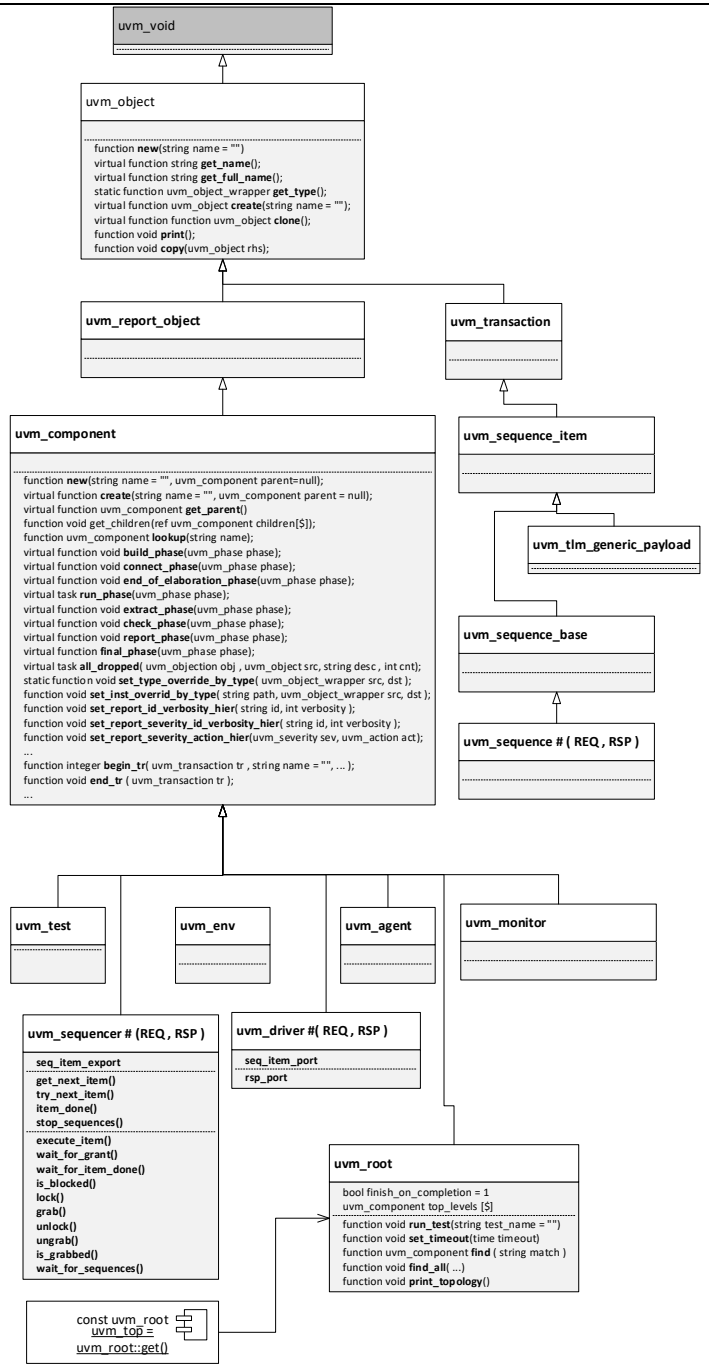
```
#include "svdpi.h"
extern void read32 (int,int*);
extern void write32(int,int);
int foo(int a) {
  read32 ( 0x8000000 , &a );
  write32( 0x8000000 , a *2 );
  // ...
}
```

SystemVerilog Code

```
import "DPI-C" context task foo(int);
export "DPI-C" write32 = task sv_write32;
export "DPI-C" task read32;
task write32(int addr,output int read_val);
 …
```

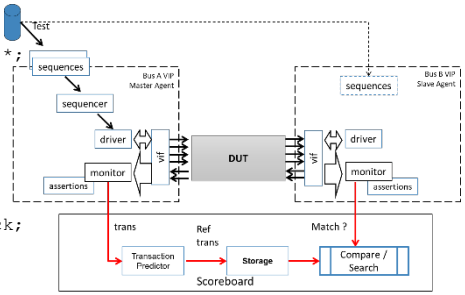
UVM Base Classes

```
class uvm_void;
endclass

class uvm_object extends uvm_void;
      //...
endclass

class uvm_transaction extends uvm_object;
      //...
endclass

class uvm_sequence_item extends uvm_transaction;
      //...
endclass

class uvm_sequence #(REQ,RSP) extends uvm_sequence_base;
      //...
endclass

class uvm_component extends uvm_report_object;
      //...
endclass

class uvm_test extends uvm_component;
      //...
endclass

class uvm_env extends uvm_component;
      //...
endclass

class uvm_agent extends uvm_component;
      //...
endclass

class uvm_monitor extends uvm_component;
      //...
endclass

class uvm_driver #(REQ,RSP) extends uvm_component;
      //...
Endclass

class uvm_sequencer#(REQ,RSP) extends uvm_component;
      //...
endclass
```



UVM TestBench

```
`timescale 1ns/100ps
`include "uvm_macros.svh"
`include "apb_if.sv"

module tb();
   import uvm_pkg::*;
   import my_verif_pkg::*;

   reg clock = 0;
   reg reset_n;

   // Generate clock
   initial
       forever begin
          clock <= ~clock;
          #25ns;
       end

   // Instantiate the Interface
   // between the DUT and the VIP
   apb_if apb_if0( clock, reset_n );

   // All other VIP interfaces
   // ...

   initial
     begin
       // Configure the VIP to use the Interface
       // VIP provider specific, see user guide
       uvm_config_db #(virtual apb_if)::
       set (      null,
                  "uvm_test_top.apb_env0.master0",
                  "vif" ,
                  apb_if0);

       // All other configurations
       // ...

       // Start UVM
       // Executed tests is specified by
       // +UVM_TESTNAME=<> in tool CLI
       run_test();
     end

   // Instantiate Design
   uart_apb dut(
     .PCLK_i    ( apb_if0.PCLK ),
     .PRESETn_i ( apb_if0.PRESETn ),
     .PADDR_i   ( apb_if0.PADDR ),
     .PPROT_i   ( apb_if0.PPROT ),
     .PSEL_i    ( apb_if0.PSEL ),
     .PENABLE_i ( apb_if0.PENABLE ),
     .PWRITE_i  ( apb_if0.PWRITE ),
     .PWDATA_i  ( apb_if0.PWDATA ),
     .PSTRB_i   ( apb_if0.PSTRB ),
     .PREADY_o  ( apb_if0.PREADY ),
     .PRDATA_o  ( apb_if0.PRDATA ),
     .PSLVERR_o ( apb_if0.PSLVERR )
    );
endmodule
```



UVM Test

```
package my_verif_pkg;
  `include "uvm_macros.svh"
  import uvm_pkg::*;
  import aed_apb_pkg::*;


  // Create a user sequence
  class my_sequence extends apb_sequence;
    `uvm_object_utils(my_sequence)
    task body();
      starting_phase.raise_objection(this);

      `uvm_info("TEST","starting...",UVM_NONE)
      #1us;
      `uvm_do_with (req , {
        req.address == 0;
        req.direction == WRITE;
      });
      #100us;

      `uvm_info("TEST","...completed",UVM_NONE)

      starting_phase.drop_objection(this);
    endtask
  endclass


  // Create a user test
  class my_verif_top_test extends uvm_test;
    `uvm_component_utils(my_verif_top_test)

    apb_env apb_env0;

    function new(string name,uvm_component parent);
      super.new(name,parent);
    endfunction

    // Build a test with the APB VIP
    function void build_phase(uvm_phase phase);

      apb_config cfg = new();

      super.build_phase(phase);

      void'(cfg.randomize() with {
        cfg.psel_width == 1;
      });

      // Configure the VIP
      // this is VIP provider specific
      uvm_config_db #(apb_config)::set( this ,
                "apb_env0.master0" , "config" , cfg );
      uvm_config_db #(integer)::set( this ,
                "apb_env0" , "nr_masters" , 1 );
        // Create the VIP instance
      apb_env0 = apb_env::type_id
                        ::create("apb_env0" , this );
    endfunction
  endclass
```

```
  // Create a test selecting the sequence
  class my_verfi_test_1 extends my_verif_top_test;
    function build_phase(uvm_phase phase);
      uvm_config_db#(uvm_object_wrapper)::set(this,
        "apb_env0.master0.sequencer.run_phase",
        "default_sequence",
        my_sequence::type_id::get());
      super.build_phase(phase);
    endfunction
  endclass
```

Scoreboard and Analysis ports

```
// Existing VIP should provide a uvm_analysis port
package apb_vip_pkg;
    // APB Monitor.
    class apb_monitor extends uvm_monitor;
        // A transfer has been completly received.
        uvm_analysis_port #(apb_transfer)
                          completed_transfer_port;

        // ...
    endclass
endpackage


// Declare the UVM Analysis Implementation Classes
`uvm_analysis_imp_decl(_apb)
`uvm_analysis_imp_decl(_tx_frame)

// Scoreboard Class
class my_scoreboard extends uvm_component;
  `uvm_component_utils(my_scoreboard)

  // UVM analysis port are used to collect transactions
  uvm_analysis_imp_apb #(apb_transfer,
                         lab_scoreboard) apb_import;
  uvm_analysis_imp_tx_frame #(uart_frame, lab_scoreboard)
                            tx_frame_import;

  int unsigned recorded_data;

  function new(  string name="lab_scoreboard",
                 uvm_component parent=null);
    super.new(name,parent);
    apb_import      = new("apb_import"     , this);
    tx_frame_import = new("tx_frame_import", this);
  endfunction

  // Record Data when a write occurs
  function void write_apb(apb_transfer trans);
    if ( (trans.direction == WRITE) &&
         (act_address == `UART_TX_RX_OR_DIVISOR_LSB))
        begin
            recorded_data = trans.data;
        end
  endfunction
```

```
  // Check data when a frame is monitored
  function void write_tx_frame(uart_frame txf);
      if (recorded_data !== uart_tx.data)
        begin
        `uvm_error("SCOREBOARD lean",
                $psprintf("mismatch (0x%h vs 0x%h)",
                recorded_data, uart_tx.data))
      end
  endfunction
endclass


/// In Test or Environment, Create and Connect Scoreboard
class my_test extends my_verif_top_test;
  `uvm_component_utils(my_test)

  lab_scoreboard my_scoreboard;

  virtual function void build_phase(uvm_phase phase);
    //..
    my_scoreboard = scbd0::type_id
                  ::create("scbd0",this);
  endfunction

  // Connect Phase
  virtual function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);

    // Connect the two port implementation to …
    // the two VIP monitors
    verif_env0.apb_env0.masters[0].monitor
       .completed_transfer_port
          .connect( scbd0.apb_import );
    verif_env0.uart_env0.agents["uart_agent0"].tx_monitor
       .completed_byte
          .connect( scbd0.tx_frame_import );
  endfunction
endclass
```

UVM Macros

Reporting

```
`uvm_info("MSG_ID", "Message" , UVM_LOW)
`uvm_error("MSG_ID", "Message" )
`uvm_fatal("MSG_ID", "Message" )
```

Register Classes to Factory

```
`uvm_component_utils( <class_name> )

`uvm_component_utils_begin( <class_name> )
  `uvm_field_int( <field> , UVM_DEFAULT | UVM_HEX )
  `uvm_field_string( <field> , UVM_DEFAULT)
`uvm_component_utils_end

`uvm_object_utils( <class_name> )

`uvm_object_utils_begin( <class_name> )
  `uvm_field_int( <field> , UVM_DEFAULT | UVM_HEX )
  `uvm_field_string( <field> , UVM_DEFAULT)
`uvm_object_utils_end
```