CHAPTER 1 CSL Testbench

TABLE 1.1 Chapter Outline

1.1 CSL Testbench Syntax and Command Summary
1.2 Testbench Commands

1.1 CSL Testbench Syntax and Command Summary

1.1.1 CSL Testbench class

The design under test (DUT) is instantiated in a testbench. Testbenches are declared as classes and are scope holders.

1.1.2 CSL Testbench class declaration

A CSL Testbench can only be declared in the global scope (like any other CSL class). The CSL Testbench class is declared as in the below example:

```
csl_testbench testbench_name {
  (objects declarations/instantiations)+
  testbench_name(){
    (testbench methods calls)+
  }
};
```

Note for example above: **bold** text represents language reserved syntax, *italics* are user defined variables.

In the testbench class' scope DUTs are instantiated. A design under test can be a CSL unit or other predefined CSL classes as shown in Table 1.2

TABLE 1.2 Rules for instantiating objects in a testbench scope

CSL class	Is instantiated in CSL Testbench scope
CSL Unit	YES
CSL Testbench	-
CSL Vector	-
CSL State Data	-
CSL Register	YES
CSL Register File	YES
CSL Fifo	YES
CSL Memory	YES
CSL Memory Map	-
CSL Memory Map Page	-
CSL ISA Element	-

TABLE 1.2 Rules for instantiating objects in a testbench scope

CSL class	Is instantiated in CSL Testbench scope
CSL ISA Field	-
CSL Field	-

1.1.3 CSL Testbench mandatory commands

The following command is mandatory to the CSL testbench class as it sets the clock signal that will be used throughout the testbench.

CSL Testbench mandatory commands

```
    add_logic(clock_generator,clock_name,period,time_base);
```

1.1.4 CSL Testbench usage and rules

Testbenches can only be declared in the global scope and cannot be instantiated anywhere as can also be seen from Table 1.3

TABLE 1.3 Vector usage rules

CSL class	Instantiates CSL Testbench
CSL Unit	-
CSL Testbench	-
CSL Vector	-
CSL State Data	-
CSL Register	-
CSL Register File	-
CSL Fifo	-
CSL Memory	-
CSL Memory Map	-
CSL Memory Map Page	-
CSL Isa Element	-
CSL Isa Field	-
CSL Field	-

Testbenches are thus a particular case of top level unit but should not be confused with the design's top level unit (a CSL design can have a top level unit and a testbench at the same time).

1.1.5 CSL Testbench commands

The following commands apply to CSL tesbench classes.

CSL Testbench commands
`set_testbench_verilog_filename(name);`
`add_logic(generate_waves, filename, wave_type);`
`add_logic(generate_report);`

1.2 Testbench Commands

add_logic(clock_generator,clock_name,period,time_base);

DESCRIPTION :

Creates a clock generator. *Clock_generator* represents the logic type to add to the testbench, *clock_name* is the name of the clock signal that is driven by the clock generator, *period* represents the period of clock signal generated by the clock_generator and *time_base* is clock period time base.
Sets the clock signal for the testbench. The clock signal must be register type.This will be used to create a clock generator that will output a clock signal on the *clock_name* signal, with the period given by the *period* parameter and the time base specified by *time_base*. The *time_base* can be one of following :

TABLE 1.4

time_base	
ps	picoseconds
fs	femtoseconds
ns	nanoseconds
us	microseconds
ms	miliseconds
s	seconds

[***CSL Testbench Syntax and Command Summary***]

EXAMPLE :

In this example a clock signal named *clk* is declared, with period 10 and time base *ps,* in the csl_testbench *tb.*

CSL CODE

```
    csl_unit dut{
      csl_port in_v(input,1), in_d(input, 8);
      csl_port out_v(output, 1), out_d(output, 8);
      csl_port clk(input);
       dut(){
    {out_v,out_d}={in_v,in_d};
    clk.set_attr(clock);}
    };
    csl_vector stim{
      stim(){
        set_unit_name ( dut );
        set_direction   ( input );
       }
    };
    csl_vector exp{
      exp(){
        set_unit_name ( dut );
        set_direction   ( output );
      }
    };
    csl_testbench tb{
      csl_signal clk(reg);
      dut dut_i;
        tb(){
      clk.set_attr(clock);
      add_logic(clock,clk,10,ps);  //create clock generator to frive DUT
      }};
```

VERILOG CODE
// DUT module

```
    module dut(in_v,
               in_d,
               out_v,
               out_d,
               clk);
      input in_v;
      input [7:0] in_d;
      input clk;
      output out_v;
      output [7:0] out_d;
     assign  {out_v,out_d}= {in_v,in_d};
      `include "dut.logic.v"
    endmodule
```

// Stimulus and expected vectors module

```
    module stim_expect_mem_template(clock,
                                    reset_,
                                    rd_en,
                                    vector_out,
                                    valid,
                                    version_err,
                                    id_err);


      parameter MEM_WIDTH = 0;
      parameter ADDR_WIDTH = 0;
      parameter VECTOR_ID = 0;
      parameter VECTOR_VERSION = 0;
```

```
  parameter VECTOR_NAME = "";
  parameter VECTOR_FILE = "";
  parameter VECTOR_RADIX = 0;
  parameter MEM_DEPTH = ((1 << ADDR_WIDTH) - 1'b1);
  input clock;
  input reset_;
  input rd_en;
  output [MEM_WIDTH - 1:0] vector_out;
  output valid;
  output version_err;
  output id_err;
  reg [MEM_WIDTH - 1:0] memory_out;
  reg [MEM_WIDTH - 1:0] stim_expect_memory[0:MEM_DEPTH] ;
  reg [ADDR_WIDTH - 1:0] rd_addr;
  reg mem_out_is_id;
  reg mem_out_is_version;
  integer mem_addr;
  reg stim_expect_memory_loaded;
  wire mem_out_is_id_or_version;
  wire mux_select;
  wire vector_id_match;
  wire vector_version_match;
  assign   mem_out_is_id_or_version = mem_out_is_id ||
mem_out_is_version;
  assign   mux_select = rd_en || ~mem_out_is_id_or_version;
  assign   vector_out = mux_select ? memory_out:{MEM_WIDTH {1'b0}};
  assign   vector_id_match = (memory_out == VECTOR_ID) & mem_out_is_id;
  assign   vector_version_match = (memory_out == VECTOR_VERSION) &
mem_out_is_version;
  assign   version_err = mem_out_is_version & memory_out !=
VECTOR_VERSION;
  assign   id_err = mem_out_is_id & memory_out != VECTOR_ID;
  assign   valid = rd_en && ~mem_out_is_id_or_version;

  always @( posedge clock or negedge reset_ )  begin
    if ( ~reset_ )  begin
      rd_addr <= {ADDR_WIDTH {1'b0}};
    end
    else if ( rd_en )  begin
        rd_addr <= rd_addr + 1;
        mem_out_is_id <= rd_addr == 0;
        mem_out_is_version <= rd_addr == 1;
      end
  end

  always @( posedge clock or negedge reset_ )  begin
    if ( rd_en )  begin
      memory_out <= stim_expect_memory[rd_addr];
    end
  end

  initial
 begin
    stim_expect_memory_loaded <= 0;
    $display("VECTOR_FILE= %s", VECTOR_FILE);
    if ( VECTOR_RADIX == 0 )  begin
      $readmemb(VECTOR_FILE, stim_expect_memory);
    end
    else  begin
      $readmemh(VECTOR_FILE, stim_expect_memory);
    end
    stim_expect_memory_loaded <= 1;
  end

  initial
 begin
    @ stim_expect_memory_loaded    ;
    if ( $test$plusargs("show_stim_expect_memory_init_state") )  begin
      $display("Initial state of vector file %s ", VECTOR_FILE);

     for (        mem_addr = 0; mem_addr < MEM_DEPTH;        mem_addr
= mem_addr + 1)  begin
        $display("mem[%d] = %x", mem_addr,
stim_expect_memory[mem_addr]);
      end
    end
  end
  endmodule
```

// Testbench module

```
  module tb();
    parameter SIM_TIMEOUT_CNT = 100;
    parameter STIM_MEM_WIDTH = 9;
    parameter STIM_ADDR_WIDTH = 0;
    parameter STIM_VECTOR_ID = 0;
    parameter STIM_VECTOR_VERSION = 0;
    parameter STIM_VECTOR_NAME = "stim";
    parameter STIM_VECTOR_FILE = "stim_output.vec";
    parameter STIM_VECTOR_RADIX = 0;
    parameter STIM_VECTOR_MAX_ERR = 0;
    parameter EXP_MEM_WIDTH = 9;
    parameter EXP_ADDR_WIDTH = 0;
    parameter EXP_VECTOR_ID = 0;
    parameter EXP_VECTOR_VERSION = 0;
    parameter EXP_VECTOR_NAME = "exp";
    parameter EXP_VECTOR_FILE = "exp_output.vec";
    parameter EXP_VECTOR_RADIX = 0;
    parameter EXP_VECTOR_MAX_ERR = 0;
    reg clk;
    reg testbench_reset;
    reg rand_valid;
    integer file_mcd;
    integer report_file_mcd;
    integer cycle_cnt;
    reg [EXP_ADDR_WIDTH - 1:0] exp_dut_i_match_count;
    reg [EXP_ADDR_WIDTH - 1:0] exp_dut_i_mismatch_count;
    reg [EXP_ADDR_WIDTH - 1:0] exp_dut_i_transaction_count;
    reg exp_dut_i_mismatch;
    wire expect_out_valid;
    wire rd_en;
    wire version_err;
    wire id_err;
    wire stop_sim = cycle_cnt >= SIM_TIMEOUT_CNT;
    wire dut_i_in_in_v;
    wire [7:0] dut_i_in_in_d;
    wire dut_i_out_out_v;
    wire dut_i_out_out_v_expect;
    wire [7:0] dut_i_out_out_d;
    wire [7:0] dut_i_out_out_d_expect;
  wire dut_i_out_out_v_mismatch_en = dut_i_out_out_v !=
dut_i_out_out_v_expect;
  wire dut_i_out_out_d_mismatch_en = dut_i_out_out_d !=
dut_i_out_out_d_expect;
  wire dut_i_out_out_v_match_en = dut_i_out_out_v ==
dut_i_out_out_v_expect;
  wire dut_i_out_out_d_match_en = dut_i_out_out_d ==
dut_i_out_out_d_expect;
  assign   rd_en = rand_valid;
  dut1 dut_i(.clk(clk),
             .in_d(dut_i_in_in_d),
             .in_v(dut_i_in_in_v),
             .out_d(dut_i_out_out_d),
             .out_v(dut_i_out_out_v));
  stim_expect_mem_template #(STIM_MEM_WIDTH,
                            STIM_ADDR_WIDTH,
                            STIM_VECTOR_ID,
                            STIM_VECTOR_VERSION,
                            STIM_VECTOR_NAME,
                            STIM_VECTOR_FILE,
                            STIM_VECTOR_RADIX)
                            stim_dut_i(.clock(clk),
                                       .id_err(id_err),
                                       .rd_en(rd_en),
                                       .reset_(testbench_reset),
                                       .valid(expect_out_valid),

.vector_out({dut_i_in_in_v,dut_i_in_in_d}),
                                       .version_err(version_err));
  stim_expect_mem_template #(EXP_MEM_WIDTH,
                            EXP_ADDR_WIDTH,
                            EXP_VECTOR_ID,
                            EXP_VECTOR_VERSION,
                            EXP_VECTOR_NAME,
                            EXP_VECTOR_FILE,
                            EXP_VECTOR_RADIX)
                            exp_dut_i(.clock(clk),
                                      .id_err(id_err),
                                      .rd_en(rd_en),
                                      .reset_(testbench_reset),
                                      .valid(expect_out_valid),
.vector_out({dut_i_out_out_v_expect,dut_i_out_out_d_expect}),
                                     .version_err(version_err));

  always @( posedge clk or negedge testbench_reset )  begin
    if ( ~testbench_reset )  begin
      cycle_cnt <= 0;
    end
    else  begin
      cycle_cnt <= cycle_cnt + 1;
    end
  end

  initial
 begin
    $system("time stamp: +20%y %m %d");
    clk = 0;
    rand_valid = 1;
    testbench_reset = 1;
    #10 testbench_reset = 0;
    #20 testbench_reset = 1;
    file_mcd = $fopen("vectors.txt");
    if ( file_mcd == 0 )  begin
      $display("Error opening vectors.txt file");
      $finish;
    end
    $display(file_mcd, "Dut outputs vs expected vectors:\n");
    report_file_mcd = $fopen("report.txt");
    $dumpfile("wavesDefaultOutputFile_dump");  //default dump file
    $dumpvars(0, tb);
    $dumpon;
    exp_dut_i_match_count = 0;
    exp_dut_i_mismatch_count = 0;
    exp_dut_i_transaction_count = 0;
  end

  always @( posedge clk )  begin
    $fdisplay(file_mcd, "dut name: %s", "dut_i", " expect vector name:
%s", "exp_dut_i", "\n");
```

```
    $fdisplay(file_mcd, "dut output: %b", dut_i_out_out_v, "expected
output: %b", dut_i_out_out_v_expect);
    $fdisplay(file_mcd, "dut output: %b", dut_i_out_out_d, "expected
output: %b", dut_i_out_out_d_expect);
  end

  always @( posedge clk )  begin
    if ( stop_sim )  begin
      $fdisplay(report_file_mcd, "Status for expect vector %s",
"exp_dut_i", "associated to dut %s", "dut_i", ":\n");
      $fdisplay(report_file_mcd, "Total number of comparisons: %b",
exp_dut_i_transaction_count, " out of which, passed: %b",
exp_dut_i_match_count, "\nOverall: %s",
exp_dut_i_mismatch_count?"failed":"passed");
    end
  end

  always #10 clk = ~clk;   //clk generator

  always @( posedge clk or negedge testbench_reset )  begin
    if ( ~testbench_reset )  begin
      exp_dut_i_mismatch_count = {EXP_ADDR_WIDTH {1'b0}};
    end
    else  begin
      if ( dut_i_out_out_v_mismatch_en )  begin
        exp_dut_i_transaction_count = exp_dut_i_transaction_count +
1'b1;
        exp_dut_i_mismatch_count = exp_dut_i_mismatch_count + 1'b1;
        $display("mismatch detected: dut %s shows value %b; evepect
vector %s shows value %b\n ", "dut_i", dut_i_out_out_v, "exp_dut_i",
dut_i_out_out_v_expect);
      end
      if ( dut_i_out_out_d_mismatch_en )  begin
        exp_dut_i_transaction_count = exp_dut_i_transaction_count +
1'b1;
        exp_dut_i_mismatch_count = exp_dut_i_mismatch_count + 1'b1;
        $display("mismatch detected: dut %s shows value %b; evepect
vector %s shows value %b\n ", "dut_i", dut_i_out_out_d, "exp_dut_i",
dut_i_out_out_d_expect);
      end
      if ( exp_dut_i_mismatch_count > EXP_VECTOR_MAX_ERR )  begin
```

```
        $display("Maximum number or errors allowed for vector %s has
been reached", "exp_dut_i");
      end
    end
  end

  always @( posedge clk or negedge testbench_reset )  begin
    if ( ~testbench_reset )  begin
      exp_dut_i_match_count = {EXP_ADDR_WIDTH {1'b0}};
    end
    else  begin
      if ( dut_i_out_out_v_match_en )  begin
        exp_dut_i_transaction_count = exp_dut_i_transaction_count +
1'b1;
        exp_dut_i_match_count = exp_dut_i_match_count + 1'b1;
       $display("match detected: dut %s shows value %b; evepect vector
%s shows value %b\n ", "dut_i", dut_i_out_out_v, "exp_dut_i",
dut_i_out_out_v_expect);
      end
      if ( dut_i_out_out_d_match_en )  begin
        exp_dut_i_transaction_count = exp_dut_i_transaction_count +
1'b1;
        exp_dut_i_match_count = exp_dut_i_match_count + 1'b1;
       $display("match detected: dut %s shows value %b; evepect vector
%s shows value %b\n ", "dut_i", dut_i_out_out_d, "exp_dut_i",
dut_i_out_out_d_expect);
      end
    end
  end
  endmodule
```

set_testbench_verilog_filename(name);

DESCRIPTION :

The default name of the Verilog module and the filename for the testbench is the name of the CSL Testbench class. This command can be used to override the default testbench name with *name*. Use csl_include to add an include file containing Verilog code to the Verilog testbench.

[***CSL Testbench Syntax and Command Summary***]

EXAMPLE :

In this example the name for the verilog file is set to “testbench”.

CSL CODE

```
csl_unit dut{
  csl_port in_v(input,1), in_d(input, 8);
  csl_port out_v(output, 1), out_d(output, 8);
  csl_port clk(input);
dut(){
clk.set_attr(clock);}
};
csl_vector stim{
  stim(){
    set_unit_name ( dut );
    set_direction ( input );
  }
};
csl_vector exp{
  exp(){
    set_unit_name ( dut );
    set_direction ( output );
}
};
csl_testbench tb{
  csl_signal clk(reg);
  dut dut_i;
  tb(){
    clk.set_attr(clock);
    add_logic(clock,clk,10,ps);
    set_testbench_verilog_filename("testbench");
  }};
```

VERILOG CODE

```
//
```

`add_logic(generate_waves``, filename, wave_type`**`);`**

DESCRIPTION :

Sets the wave generator to dump waves in an output file specified by the *filename* parameter. The wave format is specified by the wave_type parameter as shown in Table 1.5. Default is to dump waves for every signal and port from the top design below.

[***CSL Testbench Syntax and Command Summary***]

TABLE 1.5 Wave types

wave_type	generated wave format
fsdb	generates fsdb wave format
vcd	generates vcd wave format

EXAMPLE :

This example creates a dump file named *wave.dump*, which contains the generated waves with type *vcd*.

CSL CODE

```
csl_unit dut{
  csl_port in_v(input,1), in_d(input, 8);
  csl_port out_v(output, 1), out_d(output, 8);
  csl_port clk(input);
dut(){
clk.set_attr(clock);}
};
csl_vector stim{
  stim(){
    set_unit_name( dut );
    set_direction( input );
  }
};
csl_vector exp{
  exp(){
    set_unit_name( dut );
    set_direction( output );
  }
};
csl_testbench tb{
  csl_signal clk(reg);
  dut dut_i;
    tb(){
    clk.set_attr(clock);
    add_logic(clock,clk,10,ps);
```

```
        add_logic(generate_waves,"wave.vcd", vcd);
      }
    };
```

VERILOG CODE

Tb_testbench file:

```
   module tb();
     parameter SIM_TIMEOUT_CNT = 100;
     parameter STIM_MEM_WIDTH = 9;
     parameter STIM_ADDR_WIDTH = 0;
     parameter STIM_VECTOR_ID = 0;
     parameter STIM_VECTOR_VERSION = 0;
     parameter STIM_VECTOR_NAME = "stim";
     parameter STIM_VECTOR_FILE = "stim_data_out";
     parameter STIM_VECTOR_RADIX = 0;
     parameter STIM_VECTOR_MAX_ERR = 0;
     parameter EXP_MEM_WIDTH = 9;
     parameter EXP_ADDR_WIDTH = 0;
     parameter EXP_VECTOR_ID = 0;
     parameter EXP_VECTOR_VERSION = 0;
     parameter EXP_VECTOR_NAME = "exp";
     parameter EXP_VECTOR_FILE = "exp_data_out";
     parameter EXP_VECTOR_RADIX = 0;
     parameter EXP_VECTOR_MAX_ERR = 0;
     reg clk;
     reg testbench_reset;
     reg rand_valid;
     integer file_mcd;
     integer report_file_mcd;
     integer cycle_cnt;
     reg [EXP_ADDR_WIDTH - 1:0] exp_dut_i_match_count;
     reg [EXP_ADDR_WIDTH - 1:0] exp_dut_i_mismatch_count;
     reg [EXP_ADDR_WIDTH - 1:0] exp_dut_i_transaction_count;
     reg exp_dut_i_mismatch;
     wire expect_out_valid;
     wire rd_en;
     wire version_err;
     wire id_err;
     wire stop_sim = cycle_cnt >= SIM_TIMEOUT_CNT;
     wire dut_i_in_in_v;
```

```
  wire [7:0] dut_i_in_in_d;
  wire dut_i_out_out_v;
  wire dut_i_out_out_v_expect;
  wire [7:0] dut_i_out_out_d;
  wire [7:0] dut_i_out_out_d_expect;
  wire dut_i_out_out_v_mismatch_en = dut_i_out_out_v !=
dut_i_out_out_v_expect;
  wire dut_i_out_out_d_mismatch_en = dut_i_out_out_d !=
dut_i_out_out_d_expect;
  wire dut_i_out_out_v_match_en = dut_i_out_out_v ==
dut_i_out_out_v_expect;
  wire dut_i_out_out_d_match_en = dut_i_out_out_d ==
dut_i_out_out_d_expect;
  assign   rd_en = rand_valid;
  dut dut_i(.clk(clk),
            .in_d(dut_i_in_in_d),
            .in_v(dut_i_in_in_v),
            .out_d(dut_i_out_out_d),
            .out_v(dut_i_out_out_v));
  stim_expect_mem_template #(STIM_MEM_WIDTH,
                             STIM_ADDR_WIDTH,
                             STIM_VECTOR_ID,
                             STIM_VECTOR_VERSION,
                             STIM_VECTOR_NAME,
                             STIM_VECTOR_FILE,
                             STIM_VECTOR_RADIX)
                             stim_dut_i(.clock(clk),
                                        .id_err(id_err),
                                        .rd_en(rd_en),
                                        .reset_(testbench_reset),
                                        .valid(expect_out_valid),

.vector_out({dut_i_in_in_v,dut_i_in_in_d}),
                                        .version_err(version_err));
  stim_expect_mem_template #(EXP_MEM_WIDTH,
                             EXP_ADDR_WIDTH,
                             EXP_VECTOR_ID,
                             EXP_VECTOR_VERSION,
                             EXP_VECTOR_NAME,
                             EXP_VECTOR_FILE,
                             EXP_VECTOR_RADIX)
```

```
                              exp_dut_i(.clock(clk),
                                        .id_err(id_err),
                                        .rd_en(rd_en),
                                        .reset_(testbench_reset),
                                        .valid(expect_out_valid),

.vector_out({dut_i_out_out_v_expect,dut_i_out_out_d_expect}),
                                        .version_err(version_err));

  always @( posedge clk or negedge testbench_reset )  begin
    if ( ~testbench_reset )  begin
      cycle_cnt <= 0;
    end
    else  begin
      cycle_cnt <= cycle_cnt + 1;
    end
  end

  initial
 begin
   $system("time stamp: +20%y %m %d");
    clk = 0;
    rand_valid = 1;
    testbench_reset = 1;
    #10 testbench_reset = 0;
    #20 testbench_reset = 1;
    file_mcd = $fopen("vectors.txt");
    if ( file_mcd == 0 )  begin
      $display("Error opening vectors.txt file");
      $finish;
    end
    $display(file_mcd, "Dut outputs vs expected vectors:\n");
    report_file_mcd = $fopen("report.txt");
    $dumpfile("wave.dump_dump");  // create wave.dump file
    $dumpvars(0, tb);
    $dumpon;
    exp_dut_i_match_count = 0;
    exp_dut_i_mismatch_count = 0;
    exp_dut_i_transaction_count = 0;
  end
  always @( posedge clk )  begin
    $fdisplay(file_mcd, "dut name: %s", "dut_i", " expect vector name:
%s", "exp_dut_i", "\n");
    $fdisplay(file_mcd, "dut output: %b", dut_i_out_out_v, "expected
output: %b", dut_i_out_out_v_expect);
    $fdisplay(file_mcd, "dut output: %b", dut_i_out_out_d, "expected
output: %b", dut_i_out_out_d_expect);
  end

  always @( posedge clk )  begin
    if ( stop_sim )  begin
      $fdisplay(report_file_mcd, "Status for expect vector %s",
"exp_dut_i", "associated to dut %s", "dut_i", ":\n");
      $fdisplay(report_file_mcd, "Total number of comparisons: %b",
exp_dut_i_transaction_count, " out of which, passed: %b",
exp_dut_i_match_count, "\nOverall: %s",
exp_dut_i_mismatch_count?"failed":"passed");
    end
  end

  always #10 clk = ~clk;

  always @( posedge clk or negedge testbench_reset )  begin
    if ( ~testbench_reset )  begin
      exp_dut_i_mismatch_count = {EXP_ADDR_WIDTH {1'b0}};
    end
    else  begin
      if ( dut_i_out_out_v_mismatch_en )  begin
        exp_dut_i_transaction_count = exp_dut_i_transaction_count +
1'b1;
        exp_dut_i_mismatch_count = exp_dut_i_mismatch_count + 1'b1;
        $display("mismatch detected: dut %s shows value %b; evepect
vector %s shows value %b\n ", "dut_i", dut_i_out_out_v, "exp_dut_i",
dut_i_out_out_v_expect);
      end
      if ( dut_i_out_out_d_mismatch_en )  begin
        exp_dut_i_transaction_count = exp_dut_i_transaction_count +
1'b1;
        exp_dut_i_mismatch_count = exp_dut_i_mismatch_count + 1'b1;
        $display("mismatch detected: dut %s shows value %b; evepect
vector %s shows value %b\n ", "dut_i", dut_i_out_out_d, "exp_dut_i",
dut_i_out_out_d_expect);
      end
      if ( exp_dut_i_mismatch_count > EXP_VECTOR_MAX_ERR )  begin
        $display("Maximum number or errors allowed for vector %s has
been reached", "exp_dut_i");
      end
    end
  end

  always @( posedge clk or negedge testbench_reset )  begin
    if ( ~testbench_reset )  begin
      exp_dut_i_match_count = {EXP_ADDR_WIDTH {1'b0}};
    end
    else  begin
      if ( dut_i_out_out_v_match_en )  begin
        exp_dut_i_transaction_count = exp_dut_i_transaction_count +
1'b1;
        exp_dut_i_match_count = exp_dut_i_match_count + 1'b1;
       $display("match detected: dut %s shows value %b; evepect vector
%s shows value %b\n ", "dut_i", dut_i_out_out_v, "exp_dut_i",
dut_i_out_out_v_expect);
      end
      if ( dut_i_out_out_d_match_en )  begin
        exp_dut_i_transaction_count = exp_dut_i_transaction_count +
1'b1;
        exp_dut_i_match_count = exp_dut_i_match_count + 1'b1;
       $display("match detected: dut %s shows value %b; evepect vector
%s shows value %b\n ", "dut_i", dut_i_out_out_d, "exp_dut_i",
dut_i_out_out_d_expect);
      end
    end
  end
  endmodule
```

`add_logic(generate_report);`

DESCRIPTION :

This will create a report generator that will output simulation details related to testing elements on the testbench (vectors and state data) and DUT's responses.

[***CSL Testbench Syntax and Command Summary***]

EXAMPLE :

CSL CODE

```
csl_unit dut{
  csl_port in_v(input,1), in_d(input, 8);
  csl_port out_v(output, 1), out_d(output, 8);
  csl_port clk(input);
  dut(){
clk.set_attr(clock);}
};
csl_vector stim{
  stim(){
    set_unit_name (dut);
    set_direction   (input);
  }
};
csl_vector exp{
  exp(){
    set_unit_name (dut);
    set_direction   (output);
     }
};
csl_testbench tb{
  csl_signal clk(reg);
  dut dut_i;
  tb(){
    clk.set_attr(clock);
    add_logic(clock,clk,10,ps);
    add_logic(generate_report);
  }
};
```

VERILOG CODE

```
//
```