CHAPTER 1 CSL Verification Components

TABLE 1.1 Chapter Overview

1.1 CSL Verification Components Syntax and Command Summary
1.2 CSL Verification Components Commands

1.1 CSL Verification Components Syntax and Command Summary

1.1.1 Verification components classes

Verification components provide stimulus and extected results which are used by testbenches to verifiy the design under test(DUT). The verification components provide test data (stimulus vectors), compare the DUT's state at a certain time (state data) or check the data output from the DUT(expected vectors). Verification components are declared as classes and a vc scope holders. There are two main types of classes for verification components: CSL Vector and CSL State Data.

1.1.1.1 CSL Verification Components mandatory commands

```
set_unit_name(unit_name);
```

1.1.1.2 CSL Vector class

A vector is a collection of signal values captured either on a clock edge from a set of selected ports or interfaces (port containers). The actual values contained by verification components vectors are generated automatically by the C++ simulator (Csim or systemC simulator). The CSL specification is used to specify the vector's port members and to integrate with the testbench (establish connections with the proper ports and signal generators and compare units).

1.1.1.2.1 CSL Vector class declaration

A CSL Vector can only be declared in the global scope just like any other CSL class. The CSL vector class is declared as in the below example:

```
csl_vector vector_class_name {
  //no objects may be instantiated in a CSL vector
  vector_class_name(){
    (vector methods calls)+
  }
};
```

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

In the vector class' scope there aren't any objects to be specifically declared or instantiated as

shown in the table below.

TABLE 1.2 Rules for instantiating objects in the vector's scope

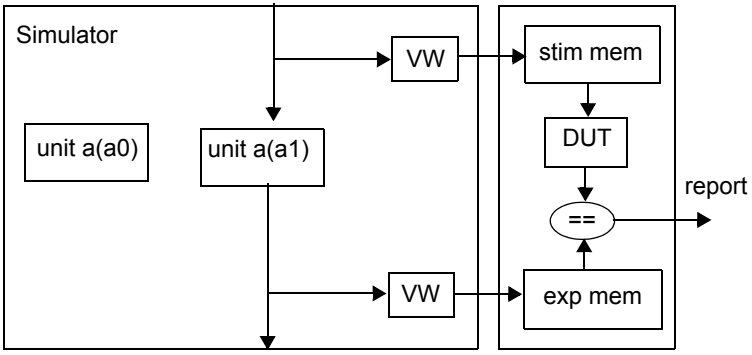
CSL class	Is instantiated in CSL Vector scope
CSL Unit	-
CSL Testbench	-
CSL Vector	-
CSL State Data	-
CSL Register	-
CSL Register File	-
CSL Fifo	-
CSL Memory Map	-
CSL Memory Map Page	-
CSL Isa Element	-
CSL Isa Field	-
CSL Field	-

However, objects are added to the vector class scope in a "non-standard" way by calling the set_unit_name(), include_only() and exclusion_list() methods. The vector methods calls (commands) are made inside the vector's constructor.

1.1.1.2.2 SystemC vectors generator

Stimulus and expected vectors used by testbench are generated by SystemC. For each unit instance is generated a set of stim and exp vectors.

FIGURE 1.1 Vectors writing flow



1.1.1.2.3 CSL Vector specific commands

The following commands are specific only to CSL Vector classes and are mandatory:

CSL Vector mandatory commands
`set_direction(direction);`

The following methods can be called on a vector class to include or exclude the ports in a vector. That is, if a vector is set to be of type stimulus (input), it will be connected to all the input ports of a unit (a DUT in the testbench) except the clock and reset ports; if the vector instance is of type expected (output) it will be connected to all the output ports of the DUT. If the user does not wish to connect all the ports to the vector, or if the user chooses to connect only one sub-interface from the output interface of a DUT to a vector, these methods are used to select the desired connection elements:

CSL Vector optional methods
`csl_vector::exclusion_list(connection_elements);`
`csl_vector::include_only(connection_element_list);`

1.1.1.2.4 CSL Vector usage and rules

Vectors are associated with a CSL unit class. It is the unit that holds the ports or interfaces that will be associated with the vector as described later in this command summary. Vectors are used in testbenches, however vector classes are not instantiated.
The rules for vector usage in other CSL classes are contained in the table below:

TABLE 1.3 Vector usage rules

CSL class	Uses CSL Vector
CSL Unit	-
CSL Testbench	YES*
CSL Vector	-
CSL State Data	-
CSL Register	-
CSL Register File	-
CSL Fifo	-
CSL Memory Map	-
CSL Memory Map Page	-
CSL Isa Element	-

TABLE 1.3 Vector usage rules

CSL class	Uses CSL Vector
CSL Isa Field	-
CSL Field	-

* but not instantiated in testbench.
Unit inputs are called stimulus vectors and unit outputs are called expected vectors.

1.1.1.3 CSL Verification components common methods

The following methods apply to both CSL Vector and State Data classes.

Common Verification Components commands

```
set_vc_header_comment(string);
//stimulus vector file example generate by SystemC
set_radix(bin|hex);
set_vc_clock(clock_signal);
set_vc_reset(reset_signal [,assertion_level]);
set_vc_stall(signal);
set_vc_valid_output_transaction(signal_expression);
set_vc_compare_trigger(signal_object | clock_signal);
set_vc_start_generation_trigger(signal);
set_vc_end_generation_trigger(signal);
set_vc_capture_edge_type(rise|fall);
set_vc_max_number_of_mismatches(numeric_expression);
set_vc_max_cycles(numeric_expression);
set_vc_output_filename(filename); //on development
add_logic(inject_stalls | inject_bubbles); //on development
```

1.2 CSL Verification Components Commands

set_unit_name(unit_name);

DESCRIPTION :

Associates the vector class with a CSL unit. This command is mandatory since it the associated unit that will determine the vector's structure. The vector has "access" to the ports inside the associated unit.

[***CSL Verification Components Syntax and Command Summary***]

EXAMPLE :

Creates a testbench named *tb* which has an instance of unit named *DUT.* The vectors *stim_vec* and *exp_vec* vectors are associated with the unit DUT.

CSL CODE

```
csl_unit DUT{
   csl_port stim_in_a0 (input);
   csl_port stim_in_a1 (input);
   csl_port exp_out_a0 (output);
   csl_port exp_out_a1 (output);
   csl_port clk(input);
   DUT(){
  clk.set_attr(clock);
   }
};
csl_vector stim_vec{
   stim_vec(){
      set_unit_name (DUT);
      set_direction   (input);
   }
};
csl_vector exp_vec{
   exp_vec(){
      set_unit_name (DUT);
      set_direction   (output);
   }
};
csl_testbench tb{
  csl_signal clk(reg);  //clock signal must be register type
  DUT dut;
  tb(){
    clk.set_attr(clock);
    add_logic(clock,clk,10,ns);
  }
};
```

VERILOG CODE

```
//DUT module
    module DUT(stim_in_a0,
               stim_in_a1,
               exp_out_a0,
               exp_out_a1,
               clk);
      input stim_in_a0;
      input stim_in_a1;
      input clk;
      output exp_out_a0;
      output exp_out_a1;
      `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 ? {MEM_WIDTH {1'b0}} : memory_out;
  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_VEC_MEM_WIDTH = 2;
    parameter STIM_VEC_ADDR_WIDTH = 0;
    parameter STIM_VEC_VECTOR_ID = 0;
    parameter STIM_VEC_VECTOR_VERSION = 0;
    parameter STIM_VEC_VECTOR_NAME = "stim_vec";
    parameter STIM_VEC_VECTOR_FILE = "stim_vec_output.vec";
    parameter STIM_VEC_VECTOR_RADIX = 0;
    parameter STIM_VEC_VECTOR_MAX_ERR = 0;
    parameter EXP_VEC_MEM_WIDTH = 2;
    parameter EXP_VEC_ADDR_WIDTH = 0;
    parameter EXP_VEC_VECTOR_ID = 0;
    parameter EXP_VEC_VECTOR_VERSION = 0;
    parameter EXP_VEC_VECTOR_NAME = "exp_vec";
```

```
  parameter EXP_VEC_VECTOR_FILE = "exp_vec_output.vec";
  parameter EXP_VEC_VECTOR_RADIX = 0;
  parameter EXP_VEC_VECTOR_MAX_ERR = 0;
  reg clk;
  reg testbench_reset;
  reg rand_valid;
  integer file_mcd;
  integer report_file_mcd;
  integer cycle_cnt;
  reg [EXP_VEC_ADDR_WIDTH - 1:0] exp_vec_dut_match_count;
  reg [EXP_VEC_ADDR_WIDTH - 1:0] exp_vec_dut_mismatch_count;
  reg [EXP_VEC_ADDR_WIDTH - 1:0] exp_vec_dut_transaction_count;
  reg exp_vec_dut_mismatch;
  wire expect_out_valid;
  wire rd_en;
  wire version_err;
  wire id_err;
  wire stop_sim = cycle_cnt >= SIM_TIMEOUT_CNT;
  wire dut_in_stim_in_a0;
  wire dut_in_stim_in_a1;
  wire dut_out_exp_out_a0;
  wire dut_out_exp_out_a0_expect;
  wire dut_out_exp_out_a1;
  wire dut_out_exp_out_a1_expect;
  wire dut_out_exp_out_a0_mismatch_en = dut_out_exp_out_a0 !=
dut_out_exp_out_a0_expect;
  wire dut_out_exp_out_a1_mismatch_en = dut_out_exp_out_a1 !=
dut_out_exp_out_a1_expect;
  wire dut_out_exp_out_a0_match_en = dut_out_exp_out_a0 ==
dut_out_exp_out_a0_expect;
  wire dut_out_exp_out_a1_match_en = dut_out_exp_out_a1 ==
dut_out_exp_out_a1_expect;
  assign   rd_en = rand_valid;
  DUT dut(.clk(clk),
          .exp_out_a0(dut_out_exp_out_a0),
          .exp_out_a1(dut_out_exp_out_a1),
          .stim_in_a0(dut_in_stim_in_a0),
          .stim_in_a1(dut_in_stim_in_a1));
  stim_expect_mem_template #(STIM_VEC_MEM_WIDTH,
                             STIM_VEC_ADDR_WIDTH,
                             STIM_VEC_VECTOR_ID,
```

```
                              STIM_VEC_VECTOR_VERSION,
                              STIM_VEC_VECTOR_NAME,
                              STIM_VEC_VECTOR_FILE,
                              STIM_VEC_VECTOR_RADIX)
                              stim_vec_dut(.clock(clk),
                                           .id_err(id_err),
                                           .rd_en(rd_en),
                                           .reset_(testbench_reset),
                                           .valid(expect_out_valid),

.vector_out({dut_in_stim_in_a0,dut_in_stim_in_a1}),
                                           .version_err(version_err));
  stim_expect_mem_template #(EXP_VEC_MEM_WIDTH,
                              EXP_VEC_ADDR_WIDTH,
                              EXP_VEC_VECTOR_ID,
                              EXP_VEC_VECTOR_VERSION,
                              EXP_VEC_VECTOR_NAME,
                              EXP_VEC_VECTOR_FILE,
                              EXP_VEC_VECTOR_RADIX)
                              exp_vec_dut(.clock(clk),
                                          .id_err(id_err),
                                          .rd_en(rd_en),
                                          .reset_(testbench_reset),
                                          .valid(expect_out_valid),

.vector_out({dut_out_exp_out_a0_expect,dut_out_exp_out_a1_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");
    $dumpvars(0, tb);
    $dumpon;
    exp_vec_dut_match_count = 0;
    exp_vec_dut_mismatch_count = 0;
    exp_vec_dut_transaction_count = 0;
  end
 always @( posedge clk )  begin
    $fdisplay(file_mcd, "dut name: %s", "dut", " expect vector name:
%s", "exp_vec_dut", "\n");
    $fdisplay(file_mcd, "dut output: %b", dut_out_exp_out_a0,
"expected output: %b", dut_out_exp_out_a0_expect);
    $fdisplay(file_mcd, "dut output: %b", dut_out_exp_out_a1,
"expected output: %b", dut_out_exp_out_a1_expect);
  end

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

  always #10 clk = ~clk;
```

```
  always @( posedge clk or negedge testbench_reset )  begin
    if ( ~testbench_reset )  begin
      exp_vec_dut_mismatch_count = {EXP_VEC_ADDR_WIDTH {1'b0}};
    end
    else  begin
      if ( dut_out_exp_out_a0_mismatch_en )  begin
       exp_vec_dut_transaction_count = exp_vec_dut_transaction_count +
1'b1;
        exp_vec_dut_mismatch_count = exp_vec_dut_mismatch_count + 1'b1;
         $display("mismatch detected: dut %s shows value %b; evepect
vector %s shows value %b\n ", "dut", dut_out_exp_out_a0,
"exp_vec_dut", dut_out_exp_out_a0_expect);
      end
      if ( dut_out_exp_out_a1_mismatch_en )  begin
       exp_vec_dut_transaction_count = exp_vec_dut_transaction_count +
1'b1;
        exp_vec_dut_mismatch_count = exp_vec_dut_mismatch_count + 1'b1;
         $display("mismatch detected: dut %s shows value %b; evepect
vector %s shows value %b\n ", "dut", dut_out_exp_out_a1,
"exp_vec_dut", dut_out_exp_out_a1_expect);
      end
     if ( exp_vec_dut_mismatch_count > EXP_VEC_VECTOR_MAX_ERR )  begin
        $display("Maximum number or errors allowed for vector %s has
been reached", "exp_vec_dut");
      end
    end
  end

  always @( posedge clk or negedge testbench_reset )  begin
    if ( ~testbench_reset )  begin
      exp_vec_dut_match_count = {EXP_VEC_ADDR_WIDTH {1'b0}};
    end
    else  begin
      if ( dut_out_exp_out_a0_match_en )  begin
       exp_vec_dut_transaction_count = exp_vec_dut_transaction_count +
1'b1;
        exp_vec_dut_match_count = exp_vec_dut_match_count + 1'b1;
       $display("match detected: dut %s shows value %b; evepect vector
%s shows value %b\n ", "dut", dut_out_exp_out_a0, "exp_vec_dut",
dut_out_exp_out_a0_expect);
      end
      if ( dut_out_exp_out_a1_match_en )  begin
```

```
       exp_vec_dut_transaction_count = exp_vec_dut_transaction_count +
1'b1;
        exp_vec_dut_match_count = exp_vec_dut_match_count + 1'b1;
       $display("match detected: dut %s shows value %b; evepect vector
%s shows value %b\n ", "dut", dut_out_exp_out_a1, "exp_vec_dut",
dut_out_exp_out_a1_expect);
      end
    end
  end
  endmodule
```

set_direction(direction);

DESCRIPTION :

This mandatory command sets the type of the vector to either stimulus or expected according to the direction parameter; thus, if the direction is set to input, the vector will be of type stimulus and if the direction is set as output, the type will be expected. It is used together with *set_unit_name (unit_name)* command, which associates the vector with a CSL unit. Once the vector is associated with a unit, the type (direction) determines the kind (direction) of ports that the vector will contain.

TABLE 1.4 Vector type according to direction parameter

vector direction	vector type
input	stimulus
output	expected

[***CSL Verification Components Syntax and Command Summary***]

EXAMPLE :

Creates two vectors *stim_vec* which is an input vector and *exp_vec* which is an output vector.

CSL CODE

```
csl_unit DUT{
   csl_port stim_in_a0 ( input );
   csl_port stim_in_a1 ( input );
   csl_port exp_out_a0 ( output );
   csl_port exp_out_a1 ( output );
   csl_port clk(input);
   DUT(){
   clk.set_attr(clock);
}
};
csl_vector stim_vec{
   stim_vec(){
      set_unit_name ( DUT);
      set_direction   ( input );
   }
};
csl_vector exp_vec{
   exp_vec(){
      set_unit_name ( DUT );
      set_direction   ( output );
   }
};
csl_testbench tb{
csl_signal clk(reg);
```

```
DUT dut;
     tb(){
clk.set_attr(clock);
add_logic(clock,clk,10,ns);
}
};
```

VERILOG CODE

Section of verilog code which shows the *set_direction()* generated code.

```
DUT dut(.clk(clk),
          .exp_out_a0(dut_out_exp_out_a0),
          .exp_out_a1(dut_out_exp_out_a1),
          .stim_in_a0(dut_in_stim_in_a0),
          .stim_in_a1(dut_in_stim_in_a1));
  stim_expect_mem_template #(STIM_VEC_MEM_WIDTH,
                             STIM_VEC_ADDR_WIDTH,
                             STIM_VEC_VECTOR_ID,
                             STIM_VEC_VECTOR_VERSION,
                             STIM_VEC_VECTOR_NAME,
                             STIM_VEC_VECTOR_FILE,
                             STIM_VEC_VECTOR_RADIX)
                             stim_vec_dut(.clock(clk),
                                          .id_err(id_err),
                                          .rd_en(rd_en),
                                          .reset_(testbench_reset),
                                          .valid(expect_out_valid),

.vector_out({dut_in_stim_in_a0,dut_in_stim_in_a1}),
                                          .version_err(version_err));
  stim_expect_mem_template #(EXP_VEC_MEM_WIDTH,
                             EXP_VEC_ADDR_WIDTH,
                             EXP_VEC_VECTOR_ID,
                             EXP_VEC_VECTOR_VERSION,
                             EXP_VEC_VECTOR_NAME,
                             EXP_VEC_VECTOR_FILE,
                             EXP_VEC_VECTOR_RADIX)
                             exp_vec_dut(.clock(clk),
                                         .id_err(id_err),
                                         .rd_en(rd_en),
                                         .reset_(testbench_reset),
```

```
                                        .valid(expect_out_valid),

.vector_out({dut_out_exp_out_a0_expect,dut_out_exp_out_a1_expect}),
                                        .version_err(version_err));
```

csl_vector::exclusion_list(connection_elements);

DESCRIPTION :

This optional command is used to exclude ports in the unit from the vector.

NOTE: the clock and reset input ports are automatically excluded from the vector if they have the CSL type clock and reset respectively. This command can be used to select the ports that will not be connected by the vector.

[***CSL Verification Components Syntax and Command Summary***]

EXAMPLE :

In this example the *stim_in_a1* port is excluded from *stim_vec_a* vector.

CSL CODE

```
csl_unit DUT_a{
   csl_port stim_in_a0 ( input );
   csl_port stim_in_a1 ( input );
   csl_port exp_out_a0 ( output );
   csl_port exp_out_a1 ( output );
   csl_port clk(input);
   DUT_a(){
   clk.set_attr(clock);   //will exclude clk from input vector
}
};
csl_vector stim_vec_a{
   stim_vec_a(){
      set_unit_name ( DUT_a );
      set_direction   ( input );
      exclusion_list  ( stim_in_a1 );
   }
};
//creates a cycle accurate vector
csl_vector exp_vec_a{
   exp_vec_a(){
      set_unit_name ( DUT_a );
      set_direction   ( output );
   }
};
csl_testbench tb{
csl_signal clk(reg);
DUT_a dut;
     tb(){
clk.set_attr(clock);
add_logic(clock,clk,10,ns);
}
```

```
};
```

VERILOG CODE
Section of verilog code which shows the effect of method.
tb_testbench.v file

```
parameter STIM_VEC_A_MEM_WIDTH = 1;
parameter EXP_VEC_A_MEM_WIDTH = 2;
DUT_a dut(.clk(clk),
          .exp_out_a0(dut_out_exp_out_a0),
          .exp_out_a1(dut_out_exp_out_a1),
          .stim_in_a0(dut_in_stim_in_a0));
 stim_expect_mem_template #(.ADDR_WIDTH(STIM_VEC_A_ADDR_WIDTH),
                            .MEM_WIDTH(STIM_VEC_A_MEM_WIDTH),
                            .VECTOR_FILE(STIM_VEC_A_VECTOR_FILE),
                            .VECTOR_ID(STIM_VEC_A_VECTOR_ID),
                            .VECTOR_NAME(STIM_VEC_A_VECTOR_NAME),
                            .VECTOR_RADIX(STIM_VEC_A_VECTOR_RADIX),
                            .VECTOR_VERSION(STIM_VEC_A_VECTOR_VERSION))
                            stim_vec_a_dut(.clock(clk),
                                           .id_err(id_err),
                                           .rd_en(rd_en),
                                           .reset_(testbench_reset),
                                           .valid(expect_out_valid),
                                        .vector_out(dut_in_stim_in_a0),
                                           .version_err(version_err));
 stim_expect_mem_template #(.ADDR_WIDTH(EXP_VEC_A_ADDR_WIDTH),
                            .MEM_WIDTH(EXP_VEC_A_MEM_WIDTH),
                            .VECTOR_FILE(EXP_VEC_A_VECTOR_FILE),
                            .VECTOR_ID(EXP_VEC_A_VECTOR_ID),
                            .VECTOR_NAME(EXP_VEC_A_VECTOR_NAME),
                            .VECTOR_RADIX(EXP_VEC_A_VECTOR_RADIX),
                            .VECTOR_VERSION(EXP_VEC_A_VECTOR_VERSION))
                            exp_vec_a_dut(.clock(clk),
                                          .id_err(id_err),
                                          .rd_en(rd_en),
                                          .reset_(testbench_reset),
                                          .valid(expect_out_valid),

.vector_out({dut_out_exp_out_a0_expect,dut_out_exp_out_a1_expect}),
                                          .version_err(version_err));
```

csl_vector::include_only(connection_element_list);

DESCRIPTION :

Creates a vector with the list of ports in the method argument. Ports must match the direction of the vector type (stimulus/input and expected/output).

[***CSL Verification Components Syntax and Command Summary***]

EXAMPLE :

In this example is included only *exp_out_a0* port in *exp_vec_a* vector.

CSL CODE

```
csl_unit DUT_a{
   csl_port stim_in_a0 ( input );
   csl_port stim_in_a1 ( input );
   csl_port exp_out_a0 ( output );
   csl_port exp_out_a1 ( output );
   csl_port clk(input);
   DUT_a(){
  clk.set_attr(clock);
}
};
csl_vector stim_vec_a{
   stim_vec_a(){
      set_unit_name ( DUT_a );
      set_direction   ( input );
   }
};
csl_vector exp_vec_a{
   exp_vec_a(){
      set_unit_name ( DUT_a );
      set_direction   ( output );
      include_only    ( exp_out_a0 );
   }
};
csl_testbench tb{
csl_signal clk(reg);
DUT_a dut;
     tb(){
clk.set_attr(clock);
add_logic(clock,clk,10,ns);
}
};
```

VERILOG CODE

Section of verilog code which shows the effect of method.

tb_testbench.v file

```
    parameter STIM_VEC_A_MEM_WIDTH = 2;
    parameter EXP_VEC_A_MEM_WIDTH = 1;
    DUT_a dut(.clk(clk),
              .exp_out_a0(dut_out_exp_out_a0),
              .stim_in_a0(dut_in_stim_in_a0),
              .stim_in_a1(dut_in_stim_in_a1));
     stim_expect_mem_template #(.ADDR_WIDTH(STIM_VEC_A_ADDR_WIDTH),
                                .MEM_WIDTH(STIM_VEC_A_MEM_WIDTH),
                                .VECTOR_FILE(STIM_VEC_A_VECTOR_FILE),
                                .VECTOR_ID(STIM_VEC_A_VECTOR_ID),
                                .VECTOR_NAME(STIM_VEC_A_VECTOR_NAME),
                                .VECTOR_RADIX(STIM_VEC_A_VECTOR_RADIX),
                                .VECTOR_VERSION(STIM_VEC_A_VECTOR_VERSION))
                                stim_vec_a_dut(.clock(clk),
                                               .id_err(id_err),
                                               .rd_en(rd_en),
                                               .reset_(testbench_reset),
                                               .valid(expect_out_valid),

   .vector_out({dut_in_stim_in_a0,dut_in_stim_in_a1}),
                                               .version_err(version_err));
      stim_expect_mem_template #(.ADDR_WIDTH(EXP_VEC_A_ADDR_WIDTH),
                                .MEM_WIDTH(EXP_VEC_A_MEM_WIDTH),
                                .VECTOR_FILE(EXP_VEC_A_VECTOR_FILE),
                                .VECTOR_ID(EXP_VEC_A_VECTOR_ID),
                                .VECTOR_NAME(EXP_VEC_A_VECTOR_NAME),
                                .VECTOR_RADIX(EXP_VEC_A_VECTOR_RADIX),
                                .VECTOR_VERSION(EXP_VEC_A_VECTOR_VERSION))
                                exp_vec_a_dut(.clock(clk),
                                              .id_err(id_err),
                                              .rd_en(rd_en),
                                              .reset_(testbench_reset),
                                              .valid(expect_out_valid),

   .vector_out(dut_out_exp_out_a0_expect),
                                              .version_err(version_err));
```

set_vc_header_comment(string**);**

DESCRIPTION :

Adds a comment to the top of the vector/state data file. The comment is ignored by the vector reader in the testbench.

[***CSL Verification Components Syntax and Command Summary***]

EXAMPLE :

FIGURE 1.2

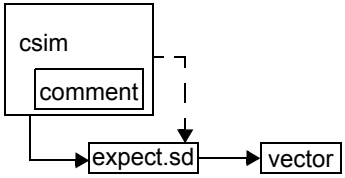
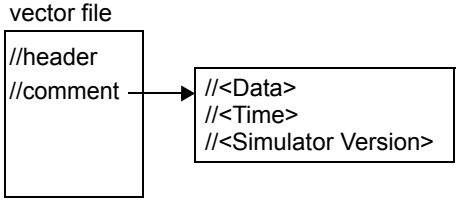


FIGURE 1.3



CSL CODE:

```
csl_unit dut{
  csl_port stim_in(input), stim_v(input), exp_d(output),exp_v(output);
  csl_port clk(input);
  dut(){
  clk.set_attr(clock);
}
};
csl_vector stim_vec{
  stim_vec(){
    set_unit_name(dut);
    set_direction(input);
    set_vc_header_comment("stimvec");
  }
};
csl_vector exp_vec{
  exp_vec(){
   set_unit_name(dut);
   set_direction(output);
}
```

```
};
csl_testbench tb{
csl_signal clk(reg);
dut dut_1(.clk(clk));
tb(){
  clk.set_attr(clock);
  add_logic(clock,clk,100,ps);
}
};
```

VERILOG CODE

```
//stimulus vector file example generate by SystemC

stim_vec_<dut instance_name>_   file content
// $var wire    1  in4  in [0:0]  $end
// $var wire    1  in3  in [0:0]  $end
// $var wire    4  in2  in [3:0]  $end
// $var wire    4  in1  in [3:0]  $end
// VEC ID:  2
// VEC VERSION:  4
1_1_1111_1111
0_1_0000_1111
1_0_1010_0101
0_0_0001_0010

//expected vector file example generated by SystemC

exp_vec_<dut_instance_name>_ file content
// $var reg    4  out4  out [3:0]  $end
// $var reg    4  out3  out [3:0]  $end
// $var reg    4  out2  out [3:0]  $end
// $var reg    4  out1  out [3:0]  $end
// VEC ID:  2
// VEC VERSION:  4
1111_1111_1111_1111
0001_0001_0001_0001
0001_0001_0001_0001
0000_0000_0000_0000
```

```
set_version(numeric_expression);
```

DESCRIPTION :

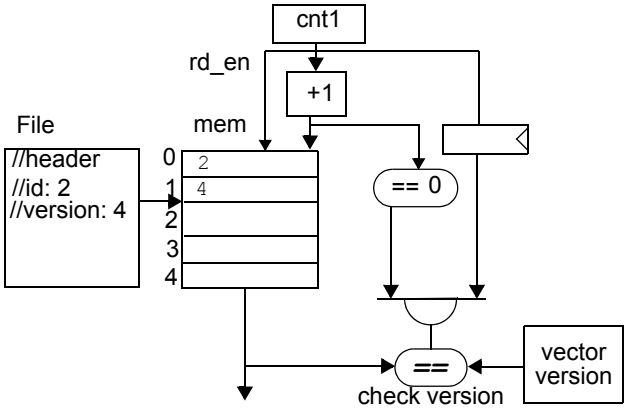
The version is inserted into the generated vector.
Version numbers are automatically assigned to verification components. This command overrides the version number for the corresponding verification component. The compiler will check for verification component number collisions. This version number will be checked against the one from generated vector/state data (from the C++ Simulator) and if a mismatch is detected the generated version checker will halt the simulation.

[***CSL Verification Components Syntax and Command Summary***]

EXAMPLE :

Sets version 2 for *stim_vec* vector.

FIGURE 1.4 The ID is added to the file header



CSL CODE:

```
csl_unit dut{
  csl_port stim_in(input), stim_v(input), exp_d(output),exp_v(output);
  csl_port clk(input);
  dut(){
  clk.set_attr(clock);
}
};
csl_vector stim_vec{
  stim_vec(){
    set_unit_name(dut);
    set_direction(input);
    set_version(2);
  }
};
```

```
csl_vector exp_vec{
  exp_vec(){
   set_unit_name(dut);
   set_direction(output);
}
};
csl_testbench tb{
csl_signal clk(reg);
dut dut_1(.clk(clk));
tb(){
  clk.set_attr(clock);
  add_logic(clock,clk,100,ps);  //add a clock generator

}
};
```

VERILOG CODE
Section of verilog code which shows the effect of method.

//vector file content

```
// VEC ID:  2
// VEC VERSION:  4
1_1_1111_1111
0_1_0000_1111
1_0_1010_0101
0_0_0001_0010
```

//testbench file content

```
module tb();
// Location of source csl unit: file name =  line number = 24
  parameter SIM_TIMEOUT_CNT = 100;
  parameter STIM_MEM_WIDTH = 1;
  parameter STIM_ADDR_WIDTH = 0;
  parameter STIM_VECTOR_ID = 0;
  parameter STIM_VECTOR_VERSION = 2;
  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 = 1;
  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;
```

set_radix(bin|hex**);**

DESCRIPTION :

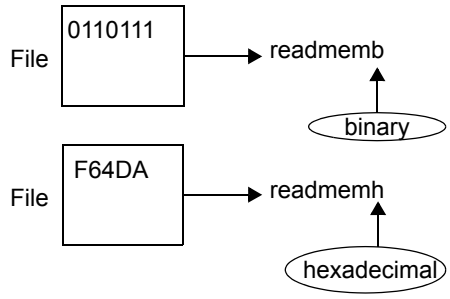
This option sets the radix for the vector format in the vector file. The default radix is binary. The vector writer will write out the vector in the radix specfied. If the vector radix is binary then the verilog testbench will use the $readmemb function. If the vector radix is hexadecimal then the testbench will use the $readmemh function.

[***CSL Verification Components Syntax and Command Summary***]

EXAMPLE :

Sets hex radix for stimulus vector.

FIGURE 1.5



CSL CODE

```
csl_unit dut{
  csl_port stim_in(input), stim_v(input), exp_d(output);
  csl_port clk(input);
dut(){
 clk.set_attr(clock);
 }
};
csl_vector stim_vec{
  stim_vec(){
    set_unit_name(dut);
    set_direction(input);
    set_radix(hex);
  }
};
csl_vector exp_vec{
  exp_vec(){
   set_unit_name(dut);
   set_direction(output);
```

```
}
};
csl_testbench tb{
csl_signal clk(reg);
dut dut_1(.clk(clk));
tb(){
  clk.set_attr(clock);
  add_logic(clock,clk,100,ps);
}
};
```

VERILOG CODE

```
//none
```

```
set_vc_clock(clock_signal);
```

DESCRIPTION :

Associates a clock with the component. If a clock is already associated with the units input ports then this command is redundant and causes an error.

[***CSL Verification Components Syntax and Command Summary***]

EXAMPLE :

Associates a clock signal *clk_s* to stimulus vector *stim_vec.*

CSL CODE

```
csl_unit dut{
  csl_port stim_in(input), stim_v(input), exp_d(output), exp_v(out-
put);
  csl_port clk(input);
  dut(){
  clk.set_attr(clock);
}
};
csl_signal clk_s;
csl_vector stim_vec{
  stim_vec(){
    set_unit_name(dut);
    set_direction(input);
    set_vc_clock(clk_s);
}
};
csl_vector exp_vec{
  exp_vec(){
   set_unit_name(dut);
   set_direction(output);
}
};
csl_testbench tb{
csl_signal clk(reg);
dut dut_1(.clk(clk));
tb(){
  clk.set_attr(clock);
  add_logic(clock,clk,100,ps);
}
};
```

VERILOG CODE

```
//
```

set_vc_reset(reset_*signal* [,assertion_level]);

DESCRIPTION :

Adds *reset_signal* to testbench object. The command *set_vc_reset* associates a reset signal with the vector. When the testbench global reset signal is active, the valid bit associated with the state data and the vectors is false. Furthermore, the state data and the vectors have their own reset signals, so they can be reset even if the global reset signal is inactive. The *assertion_level* for reset defaults to low true. The *assertion_level* can be optionally set to high true.

This command specifies the signal that controls when to set the vector valid bit to true. This command tells the vector generator to start writing valid vectors after reset. The typical signal to use is reset which is the default start vector control signal. Many designs generate garbage prior to reset and generating valid vectors prior to reset is useless for comparison purposes.

Reset_signal has to be connected to one of the testbench signals. It is illegal to connect the reset_signal to any signal declared outside the testbench scope.

There is one testbench reset signal which drives the testbench, the stimulus and expected vectors and the DUT.

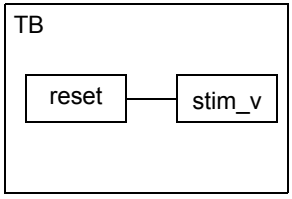
Stim vectors are read from the testbench memory after the reset signal is asserted. Stim vectors are captured by the csim, after csim reset.

[***CSL Verification Components Syntax and Command Summary***]

EXAMPLE :

Associate a reset signal *r* to stimulus vector *stim_vec.*

FIGURE 1.6 Connect reset to stimulus vector



CSL CODE

```
csl_unit dut{
  csl_port stim_in(input), stim_v(input), exp_d(output), exp_v(out-
put);
  csl_port clk(input);
  dut(){
  clk.set_attr(clock);
}
};
csl_signal r;
csl_vector stim_vec{
  stim_vec(){
    set_unit_name(dut);
    set_direction(input);
    set_vc_reset(r);
```

```
      }
};
csl_vector exp_vec{
  exp_vec(){
   set_unit_name(dut);
   set_direction(output);
}
};
csl_testbench tb{
csl_signal clk(reg);
dut dut_1(.clk(clk));
tb(){
  clk.set_attr(clock);
  add_logic(clock,clk,100,ps);
}
};
```

VERILOG CODE

set_vc_stall(signal**);**

DESCRIPTION :

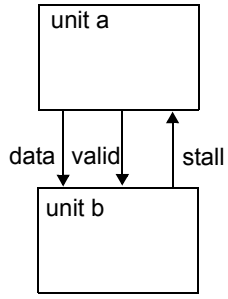
Associates a units stall signal with a stimulus vector. After reset is deasserted and the optional triggers are asserted, the vector or state is captured when the stall signal is off.

[***CSL Verification Components Syntax and Command Summary***]

EXAMPLE :

The stimulus vector driving *unit b* can be stalled with signal stall which is an output of *unit b* if specified.

FIGURE 1.7



CSL CODE

```
csl_unit dut{
  csl_port stim_in(input), stim_v(input), exp_d(output), exp_v(out-
put), stall(output);
  csl_port clk(input);
  dut(){
  clk.set_attr(clock);
}
};
csl_vector stim_vec{
  stim_vec(){
    set_unit_name(dut);
    set_direction(input);
    set_vc_stall(stall);
  }
};
csl_vector exp_vec{
  exp_vec(){
   set_unit_name(dut);
   set_direction(output);
}
```

```
};
csl_testbench tb{
csl_signal clk(reg);
dut dut_1(.clk(clk));
tb(){
  clk.set_attr(clock);
  add_logic(clock,clk,100,ps);
}
};
```

VERILOG CODE

```
//
```

set_vc_valid_output_transaction(signal_expression);

DESCRIPTION :

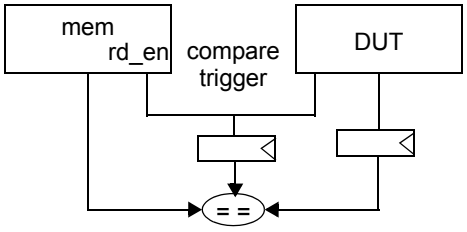
The *set_vc_valid_output_transaction()* command is used to set the transaction. *Signal_expression* indicates that the output transaction is valid. Thus, if the expression is a clock, than the transaction is always valid (valid = 1); if the expression is a signal, then the transaction is valid when the signal is true (valid = signal); if the transaction is a signal expression than the transaction is valid when the signal expression is true (e.g. valid = x | y | z). This command applies only to output vectors or state data.

The expression controls when the expected vector or state data is captured by C++ simulator and when comparisons are made in the testbench between the dut output and the expected vector.

[***CSL Verification Components Syntax and Command Summary***]

EXAMPLE :

FIGURE 1.8



CSL CODE

```
csl_unit dut{
  csl_port stim_in(input), stim_v(input), exp_d(output),
exp_v(output);
  csl_port clk(input);
  dut(){
  clk.set_attr(clock);
}
};
csl_signal r(wire);
csl_vector stim_vec{
  stim_vec(){
    set_unit_name(dut);
    set_direction(input);
    set_vc_valid_output_transactions(r);
  }
};
csl_vector exp_vec{
  exp_vec(){
   set_unit_name(dut);
   set_direction(output);
```

```
}
};
csl_testbench tb{
csl_signal clk(reg);
dut dut_1(.clk(clk));
tb(){
  clk.set_attr(clock);
  add_logic(clock,clk,100,ps);
}
};
```

VERILOG CODE

//

set_vc_compare_trigger(signal_object | clock_signal);

DESCRIPTION :

The vc trigger is a signal, a signal expression or a clock signal. The expected vector type is inferred from the compare trigger type. If the compare trigger is a clock then the expected vector will be compared to the DUT outputs each cycle. If it is a signal or a signal expression then the vector type is an transaction accurate.Compare trigger is only applied on expected vector.

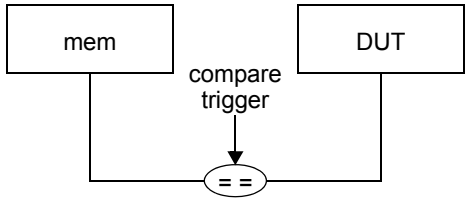
The signal indicates that a compare should be performed between the expected vc and DUT vc where vc is either a vector or state data.

The comparison unit that compares vectors of this type uses either a clock signal or an event object to perform the comparisons. If the vector type is an transaction accurate then an event based on combinational signals is specified (signal expression). If the vector type is an cycle accurate then a clock signal triggers the comparison.

[***CSL Verification Components Syntax and Command Summary***]

EXAMPLE :

FIGURE 1.9



CSL CODE

```
csl_unit dut{
  csl_port stim_in(input), stim_v(input), exp_d(output),
exp_v(output);
  csl_port clk(input);
  dut(){
  clk.set_attr(clock);
}
};
csl_vector stim_vec{
  stim_vec(){
    set_unit_name(dut);
    set_direction(input);
  }
};
csl_signal en;
csl_signal valid;
csl_vector exp_vec{
  exp_vec(){
   set_unit_name(dut);
```

```
   set_direction(output);
set_vc_compare_trigger( en & valid);
}
};
csl_testbench tb{
csl_signal clk(reg);
dut dut_1(.clk(clk));
tb(){
  clk.set_attr(clock);
  add_logic(clock,clk,100,ps);
}
};
```

VERILOG CODE

```
//
```

set_vc_start_generation_trigger(*signal***);**

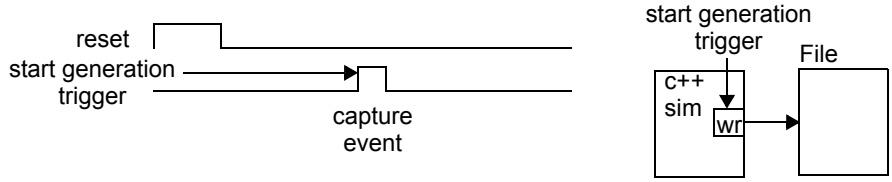
DESCRIPTION :

Name of signal that triggers the capture of the vector or state data.
Capture or compare the vector or state data on either an event or a clock edge.
Overrides the control of start state data generation by reset. Instead, start generating state data after event occurs. This command tells the state data generator to start writing state data when the compare trigger set by set_vc_start_compare_trigger() is asserted . When the compare trigger set by *set_vc_compare _trigger()* is asserted.This command is used to control when to start writing state data. A typical signal to use for event is reset which is the default start state data control signal. This command overrides verification components set as the start generation trigger. If reset is asseciated with the vector or state data, this command is mandatory. This command overrides reset as the start generation trigger. if the reset is not associated with the vector or state data, this command is mandatory. Many designs generate garbage prior to reset and generating state data prior to reset is useless for comparison purposes.

[***CSL Verification Components Syntax and Command Summary***]

EXAMPLE :

FIGURE 1.10



CSL CODE

```
csl_unit dut{
  csl_port stim_in(input), stim_v(input), out_d(output),
out_v(output);
  csl_port clk(input);
dut(){
  clk.set_attr(clock);
}
};
csl_signal trigg;
csl_vector stim_vec{
  stim_vec(){
    set_unit_name(dut);
    set_direction(input);
    set_vc_start_generation_trigger(trigg);
  }
};
csl_vector exp_vec{
  exp_vec(){
```

```
    set_unit_name(dut);
    set_direction(output);
}
};
csl_testbench tb{
csl_signal clk(reg);
dut dut_1(.clk(clk));
tb(){
  clk.set_attr(clock);
  add_logic(clock,clk,100,ps);
}
};
```

VERILOG

none

set_vc_end_generation_trigger(signal**);**

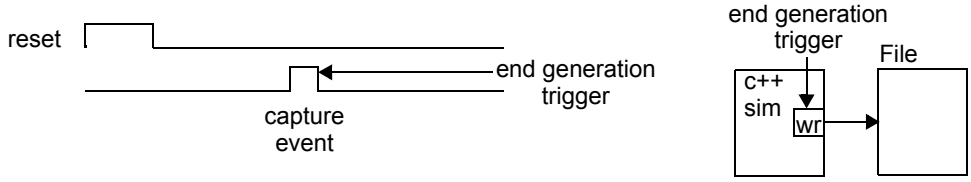
DESCRIPTION :

Name of signal that stops the vector or state data recording, if this is not set by the user then the cslc stops collecting vectors/state data when the C++ simulator stops.

[***CSL Verification Components Syntax and Command Summary***]

EXAMPLE :

FIGURE 1.11



CSL CODE

```
csl_unit dut{
  csl_port stim_in(input), stim_v(input), exp_out(output),
exp_v(output);
  csl_port clk(input);
  dut(){
 clk.set_attr(clock);
}
};
csl_signal trigg;
csl_vector stim_vec{
  stim_vec(){
    set_unit_name(dut);
    set_direction(input);
    set_vc_end_generation_trigger(trigg);
  }
};
csl_vector exp_vec{
  exp_vec(){
   set_unit_name(dut);
   set_direction(output);
}
};
csl_testbench tb{
csl_signal clk(reg);
dut dut_1(.clk(clk));
tb(){
  clk.set_attr(clock);
```

```
  add_logic(clock,clk,100,ps);
}
};
```

VERILOG CODE

```
none
```

set_vc_capture_edge_type(rise|fall);

DESCRIPTION :

Captures the vector or state data on the rising or falling edge of the capture expression (clock, signal, expression).

[***CSL Verification Components Syntax and Command Summary***]

EXAMPLE :

Sets the capture on the rise edge. The default is rising edge.

CSL CODE

```
csl_unit dut{
  csl_port stim_in(input), stim_v(input),exp_out(output),
exp_v(output);
  csl_port clk(input);
  dut(){
  clk.set_attr(clock);
}
};
csl_vector stim_vec{
  stim_vec(){
    set_unit_name(dut);
    set_direction(input);
    set_vc_capture_edge_type(rise);
  }
};
csl_vector exp_vec{
  exp_vec(){
   set_unit_name(dut);
   set_direction(output);
}
};
csl_testbench tb{
csl_signal clk(reg);
dut dut_1(.clk(clk));
tb(){
  clk.set_attr(clock);
  add_logic(clock,clk,100,ps);
}
};
```

VERILOG CODE

```
none
```

set_vc_max_number_of_mismatches(numeric_expression**);**

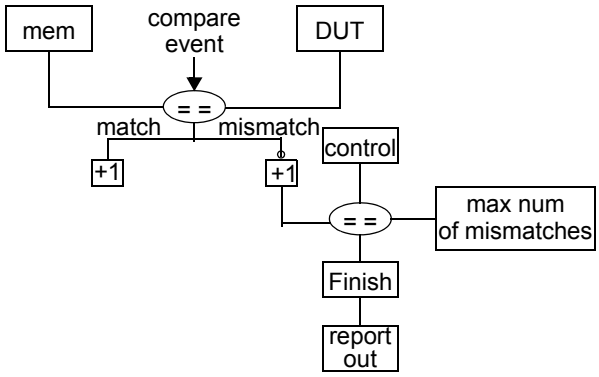
DESCRIPTION :

This command stops the state data or vector comparisons after the max number of mismatches is reached. The vector comparator in the testbench counts the number of mismatches between the expected vector and the DUT generated vector. When the error count equals the max mismatch count then the simulation stops.

[***CSL Verification Components Syntax and Command Summary***]

EXAMPLE :

FIGURE 1.12



CSL CODE

```
csl_unit dut{
  csl_port stim_in(input), stim_v(input), exp_out(output), exp_v(out-
put);
  csl_port clk(input);
  dut(){
  clk.set_attr(clock);
}
};
csl_vector stim_vec{
  stim_vec(){
    set_unit_name(dut);
    set_direction(input);
    set_vc_max_number_of_mismatches(5);
  }
};
csl_vector exp_vec{
  exp_vec(){
   set_unit_name(dut);
   set_direction(output);
```

```
}
};
csl_testbench tb{
csl_signal clk(reg);
dut dut_1(.clk(clk));
tb(){
  clk.set_attr(clock);
  add_logic(clock,clk,100,ps);

}
};
```

VERILOG CODE

Section of verilog code which shows the effect of method.

```
module tb();
// Location of source csl unit: file name =  line number = 22
  parameter SIM_TIMEOUT_CNT = 100;
  parameter STIM_VEC_MEM_WIDTH = 2;
  parameter STIM_VEC_ADDR_WIDTH = 0;
  parameter STIM_VEC_VECTOR_ID = 0;
  parameter STIM_VEC_VECTOR_VERSION = 0;
  parameter STIM_VEC_VECTOR_NAME = "stim_vec";
  parameter STIM_VEC_VECTOR_FILE = "stim_vec_output.vec";
  parameter STIM_VEC_VECTOR_RADIX = 0;
  parameter STIM_VEC_VECTOR_MAX_ERR = 5;
  parameter EXP_VEC_MEM_WIDTH = 2;
  parameter EXP_VEC_ADDR_WIDTH = 0;
  parameter EXP_VEC_VECTOR_ID = 0;
  parameter EXP_VEC_VECTOR_VERSION = 0;
  parameter EXP_VEC_VECTOR_NAME = "exp_vec";
  parameter EXP_VEC_VECTOR_FILE = "exp_vec_output.vec";
  parameter EXP_VEC_VECTOR_RADIX = 0;
  parameter EXP_VEC_VECTOR_MAX_ERR = 0;
```

set_vc_max_cycles(numeric_expression**);**

DESCRIPTION :

Sets the number of cycles to time out after the last event. If the time out is 500 then if 500 cycles have elapsed since the last vc event, then the simulation stops. It is an error to use this command with a cycle accurate vector (clock event).

[***CSL Verification Components Syntax and Command Summary***]

EXAMPLE :

CSL CODE

```
 csl_unit dut{
  csl_port stim_i(input),exp_o(output);
  csl_port clk(input);
  dut(){
clk.set_attr(clock);
 }
};
csl_signal trigg(wire);
csl_vector stim{
  stim(){
    set_unit_name(dut);
    set_direction(input);
    set_vc_start_generation_trigger(trigg);
    set_vc_max_cycles(100);
  }
};
csl_vector exp{
  exp(){
    set_unit_name(dut);
    set_direction(output);
 }
};
csl_testbench tb{
csl_signal clk(reg);
dut dut_1(.clk(clk));
tb(){
  clk.set_attr(clock);
  add_logic(clock,clk,100,ps);
}
};
```

VERILOG CODE

set_vc_output_filename(filename**);**

DESCRIPTION :

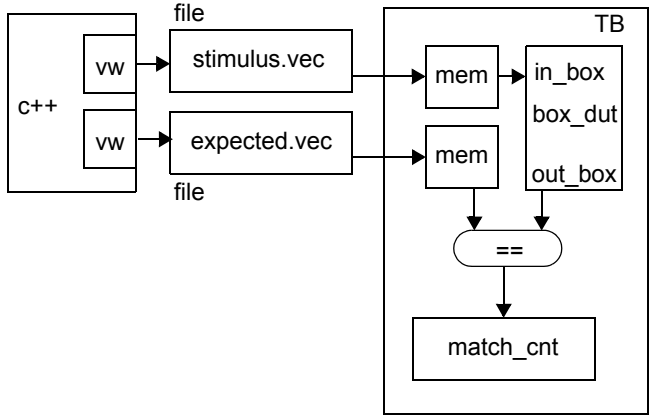
The name of the file to write the vector or state data to.

Sets the output filename. The testbench will write the result of the comparison between the vectors from the DUT output and the expected vector in *filename*. This command is useful if the user wants to see the DUT output and expected vectors, stacked on top of one another in ASCII format. Output is written in the radix of the verification components.

[***CSL Verification Components Syntax and Command Summary***]

EXAMPLE :

FIGURE 1.13



CSL CODE:

```
csl_unit dut{
  csl_port  exp_out(output),exp_v(output), stim_in(input),
stim_v(input);
  csl_port clk(input);
  dut() {
   clk.set_attr(clock);
 }
};
csl_vector stim_vec{
  stim_vec(){
  set_unit_name(dut);
  set_direction(input);
  }
};
csl_vector exp_vec{
  exp_vec(){
```

```
    set_unit_name(dut);
    set_direction(output);
    set_vc_output_filename("expected");
  }
};
csl_testbench tb{
  csl_signal clk(reg);
  dut dut;
  tb(){
  clk.set_attr(clock);
  add_logic(clock,clk,10,ns);
}
 };
```

VERILOG CODE

none

```
add_logic(inject_stalls | inject_bubbles);
```

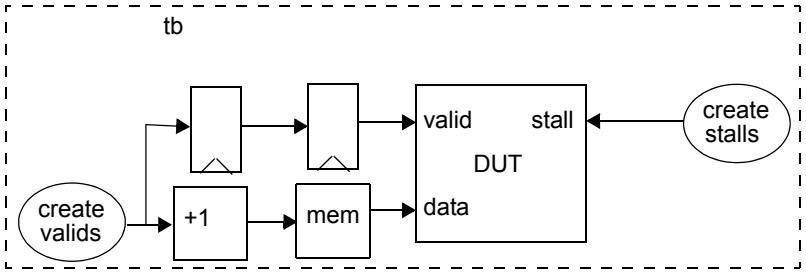
DESCRIPTION :

Inject stalls or bubbles in the testbench. If the stall signal is set, the valid signal is reset and the data is invalid. If this is not set then the stall input is tied to 0.

[***CSL Verification Components Syntax and Command Summary***]

EXAMPLE :

FIGURE 1.14 Inject stalls



CSL CODE

```
csl_unit dut{
  csl_port stim_in(input), exp_out(output);
  csl_port clk(input);
  dut(){
 clk.set_attr(clock);
}
};
csl_vector stim_vec{
  stim_vec(){
    set_unit_name(dut);
    set_direction(input);
 }
};
csl_vector exp_vec{
  exp_vec(){
    set_unit_name(dut);
    set_direction(output);
    add_logic(inject_stalls);
}
};
csl_testbench tb{
  csl_signal clk(reg);
  dut dut;
  tb(){
```

```
    clk.set_attr(clock);
    add_logic(clock,clk,10,ns);
  }
  };
```

VERILOG CODE

```
  //
```