```
vc_name.set_connection_type(connection_type); - for release 2.0
//connection_type: enum: FILE_SOCKET - for release 2.0
vector_name.add_signal(signal_object);//for release 2.0
vector_name.add_signal_group(signal_group_name); //for releaase 2.0
vector_name.insert_random_pipeline_bubble(); //move to vc as v2.0
vector_name.insert_random_stalls(); //move to vc as v2.0
```

1.1 Gets methods :

```
int vc_name.get_vc_version();
string vc_name.get_vc_header_comment();
enum vc_name.get_radix();
vc_name.get_vc_unit();
signal vc_name.get_vc_clock();
signal vc_name.get_vc_reset();
signal vc_name.get_vc_stall();
signal vc_name.get_vc_valid_output_transaction();
signal vc_name.get_vc_start_generation_trigger();
signal vc_name.get_vc_end_generation_trigger();
enum vc_name.get_vc_capture_edge_type();
int vc_name.get_vc_max_number_of_valid_transactions();
int vc_name.get_vc_max_number_of_mismatches();
int vc_name.get_vc_timeout();
string vc_name.get_output_filename();
```

1.2 CSL State Data

1.2.1 CSL State Data class

State data is a collection of the values in a state element. A state element can be a register, register

file, fifo or a memory. State data is captured periodically when an associated transaction event occurs. For example, a register file changes state when there is a write enable. The state data file is loaded into the testbench state data memory. Each RTL DUT state data transation is compared against the state data expected results stored in the state data memory. The state data can be regarded as a collection of snapshots of the state element at different moments in time. The actual values contained in the state data snapshot are generated automatically by the C++ simulator (Csim). The CSL specification is used to "tune" state data's settings and to integrate it inside the test-bench (establish connections with the proper ports and signal generators and state compare units).

1.2.1.1 CSL State Data declaration

A CSL State Data can only be declared in the global scope just like any other CSL class. The CSL state data class is declared as in the below example:

```
csl_state_data state_data_class_name {
  //no objects may be instantiated in a CSL vector
  state_data_class_name(){
    (state data methods calls)+
  }
};
```

Note for example above: **bold** text represents language reserved syntax, *italics* are user defined variables, green commented text details language notes and blue text is a short BNF representation of CSL commands/declarations.

In the state data class' scope there aren't any objects to be specifically declared or instantiated as shown in the table below.

TABLE 1.1 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	-

1.2.1.1.1 CSL State Data usage and rules

State data is associated with a memory instance. A memory instance can be an instance of a register, register file, fifo or a simple memory. State data is used in testbenches however state data classes are not instantiated.
The rules for state data usage are contained in the table below:

TABLE 1.2 Vector usage rules

CSL class	Uses CSL State Data
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	-
CSL Isa Field	-
CSL Field	-

1.2.1.1.2 CSL State Data specific methods

1.3 The following command, specific only to CSL State Data classes, is also mandatory.

CSL State Data mandatory commands

```
set_mem_instance_name(memory_instance_name);
set_vc_unit_name(unit_name);
set_vc_clock(signal);
```

1.4 Verification Components methods

set_vc_max_number_of_valid_transactions(*numeric_expression***);**

DESCRIPTION :

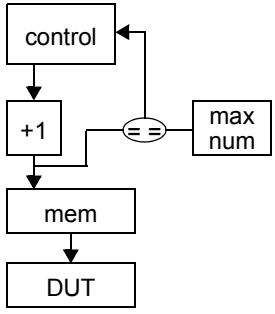
Stop capturing events when maximum number of capture events is reached
The verification transaction are writen to the output file until the maximum number of capture events is reached . After the maximum transact.ion count is reached the C++ vector writer stops writing verfication transactions to the output file or stream.

Stimulus Verification Transaction Counter
The testbench stimulus vector reader counts the number of verification transactions that are read from the stimulus verification transaction input stream or memory and when the transaction count equals max number of vectors no further verification transactions are read and stimulus verification transaction flag is set. The testbench prints a message specifying that the maximum number of stimulus verification transactions has been read.

Expected Verification Transaction Counter
The testbench stimulus vector reader counts the number of verification transactions that are read from the expected verification transaction input stream or memory and when the transaction count equals max number of vectors no further verification transactions are read and expected verification transaction flag is set. The testbench prints a message specifying that the maximum number of expected verification transactions has been read.

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

FIGURE 1.1 Maximum number of vectors



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_valid_transactions(10);
  }
};
csl_vector exp_vec{
  exp_vec(){
   set_unit_name(dut);
   set_direction(output);
}
};
csl_testbench tb{
csl_signal clk(wire);
dut dut_1(.clk(clk));
tb(){
  clk.set_attr(clock);
  add_logic(clock,clk,100,ps);

}
};
```

VERILOG CODE

int *vc_name*.get_vc_version();

DESCRIPTION :

Return the unique id number for the specified verification component.

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

EXAMPLE :

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{
  exp_vec(){
    set_unit_name(dut);
    set_direction(output);
    set_version(2);
  }
};
csl_vector exp_vec{
  stim_vec(){
   set_unit_name(dut);
    set_direction(output);
    set_version(exp_vec.get_version());
  }
};
csl_testbench tb{
  csl_signal clk;
  dut dut;
  tb(){
  clk.set_attr(clock);
  add_logic(clock,clk,10,ns);
}
 };
```

```
string vc_name.get_vc_header_comment();
```

DESCRIPTION :

Get the comment from the top of the vector/state data file.

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

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_header_comment("stimvec");
  }
};
csl_vector exp_vec{
exp_vec(){
   set_unit_name(dut);
    set_direction(output);
    set_vc_header_comment(stim_vec.get_vc_header_comment());
}
};
csl_testbench tb{
  csl_signal clk;
  dut dut;
  tb(){
  clk.set_attr(clock);
  add_logic(clock,clk,10,ns);
}
 };
```

VERILOG CODE

```
//
```

enum *vc_name*.**get_radix();**

DESCRIPTION :

Get the radix for the vector format.

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

EXAMPLE :

//

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_radix(hex);
  }
};
csl_vector exp_vec{
  exp_vec(){
    set_unit_name(dut);
    set_direction(output);
    set_radix(stim_vec.get_radix());
  }
};
csl_testbench tb{
  csl_signal clk;
  dut dut;
  tb(){
  clk.set_attr(clock);
  add_logic(clock,clk,10,ns);
}
 };
```

VERILOG CODE

```
//
```

vc_name.**get_vc_unit();**

DESCRIPTION :

Get the name of the module that the vector or state data element is associated with.

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

EXAMPLE :

//

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);
  }
};
csl_vector exp_vec{
  exp_vec(){
    set_unit_name(stim_vec.get_vc_unit());
    set_direction(output);
  }
};
csl_testbench tb{
  csl_signal clk;
  dut dut;
  tb(){
  clk.set_attr(clock);
  add_logic(clock,clk,10,ns);
}
 };
```

VERILOG CODE

```
//
```

```
signal vc_name.get_vc_clock();
```

DESCRIPTION :

Get the name of clock that triggers the capture of the vector or state data.

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

EXAMPLE :

//

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 clk(1);
csl_vector stim_vec{
  stim_vec(){
    set_unit_name(dut);
    set_direction(input);
    set_vc_clock(clk);
  }
};
csl_vector exp_vec{
  exp_vec(){
    set_unit_name(dut);
    set_direction(output);
    set_vc_clock(stim_vec.get_vc_clock());
  }
};
csl_testbench tb{
  csl_signal clk;
  dut dut;
  tb(){
  clk.set_attr(clock);
  add_logic(clock,clk,10,ns);
}
 };
```

VERILOG CODE

```
//
```

```
signal vc_name.get_vc_reset();
```

DESCRIPTION :

Get *signal_name* reset signal from *testbench_object_name*.

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

EXAMPLE :

CSL CODE

```
csl_unit dut{
  csl_port stim_in(input),stim_v(input),exp_out(output),exp_v(output);
  csl_port clk(clock);
dut(){
  clk.set_attr(clock);}
};
csl_signal res(1);
csl_vector stim_vec{
  stim_vec(){
    set_unit_name(dut);
    set_direction(input);
    set_vc_reset(res);
  }
};
csl_vector exp_vec{
  exp_vec(){
    set_unit_name(dut);
    set_direction(output);
    set_vc_reset(stim_vec.get_vc_reset());
  }
};
csl_testbench tb{
  csl_signal clk;
  dut dut;
  tb(){
  clk.set_attr(clock);
  add_logic(clock,clk,10,ns);
}
 };
```

VERILOG CODE

```
//
```

signal *vc_name*.**get_vc_stall();**

DESCRIPTION :

Get the name of signal that triggers the capture of the vector or state data;

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

EXAMPLE :

//

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 st(1);
csl_vector stim_vec{
  stim_vec(){
    set_unit_name(dut);
    set_direction(input);
    set_vc_stall(st);
  }
};
csl_vector exp_vec{
  exp_vec(){
    set_unit_name(dut);
    set_direction(output);
    set_vc_stall(stim_vec.get_vc_stall());
  }
};
csl_testbench tb{
  csl_signal clk;
  dut dut;
  tb(){
  clk.set_attr(clock);
  add_logic(clock,clk,10,ns);
}
 };
```

VERILOG CODE

```
//
```

```
signal vc_name.get_vc_valid_output_transaction();
```

DESCRIPTION :

Get the clock signal or an event object uses to perform the comparisons.

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

EXAMPLE :

//

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_output_transaction(10);
  }
};
csl_vector exp_vec{
  exp_vec(){
    set_unit_name(dut);
    set_direction(output);
    set_vc_output_transaction(stim_vec.get_vc_output_transaction());
  }
};
csl_testbench tb{
  csl_signal clk;
  dut dut;
  tb(){
  clk.set_attr(clock);
  add_logic(clock,clk,10,ns);
}
 };
```

VERILOG CODE

```
//
```

signal vc_name.**get_vc_start_generation_trigger();**

DESCRIPTION :

Get the command used to control when to start writing state datas.

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

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_start_generation_trigger(trigg);
  }
};
csl_vector exp_vec{
  exp_vec(){
    set_unit_name(dut);
    set_direction(output);
    set_vc_start_generation_trigger(
              stim_vec.get_vc_start_generation_trigger());
  }
};
csl_testbench tb{
  csl_signal clk;
  dut dut;
  tb(){
  clk.set_attr(clock);
  add_logic(clock,clk,10,ns);
}
 };
```

VERILOG CODE

```
//
```

```
signal vc_name.get_vc_end_generation_trigger();
```

DESCRIPTION :

Get the event that stops the vector or state data recording ;

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

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);
    set_vc_end_generation_trigger(
              stim_vec.get_vc_end_generation_trigger());
  }
};
csl_testbench tb{
  csl_signal clk;
  dut dut;
  tb(){
  clk.set_attr(clock);
  add_logic(clock,clk,10,ns);
}
 };
```

VERILOG CODE

```
//
```

```
enum vc_name.get_vc_capture_edge_type();
```

DESCRIPTION :

Get the capture edge type rise or fall of clock of the vector or state data.

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

EXAMPLE :

//

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);
    set_vc_capture_edge_type(
              stim_vec.get_vc_capture_edge_type());
  }
};
csl_testbench tb{
  csl_signal clk;
  dut dut;
  tb(){
  clk.set_attr(clock);
  add_logic(clock,clk,10,ns);
}
 };
```

VERILOG CODE

```
//
```

```
int vc_name.get_vc_max_number_of_valid_transactions();
```

DESCRIPTION :

Get the maximum number of captured events.

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

EXAMPLE :

//

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_max_number_of_valid_transactions(10);
  }
};
csl_vector exp_vec{
  exp_vec(){
    set_unit_name(dut);
    set_direction(output);
    set_vc_max_number_of_valid_transactions(
              stim_vec.get_vc_max_number_of_valid_transactions());
  }
};
csl_testbench tb{
  csl_signal clk;
  dut dut;
  tb(){
  clk.set_attr(clock);
  add_logic(clock,clk,10,ns);
}
 };
```

VERILOG CODE

```
//
```

```
int vc_name.get_vc_max_number_of_mismatches();
```

DESCRIPTION :

Return the max number of mismatches seted for vector.

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

EXAMPLE :

//

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_max_number_of_mismatches(10);
  }
};
csl_vector exp_vec{
  exp_vec(){
    set_unit_name(dut);
    set_direction(output);
    set_vc_max_number_of_mismatches(
             stim_vec.get_vc_max_number_of_mismatches());
  }
};
csl_testbench tb{
  csl_signal clk;
  dut dut;
  tb(){
  clk.set_attr(clock);
  add_logic(clock,clk,10,ns);
}
 };
```

VERILOG CODE

```
//
```

```
int vc_name.get_vc_timeout();
```

DESCRIPTION :

Get the number of cycles to time out after the last event.

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

EXAMPLE :

//

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_timeout(10);
  }
};
csl_vector exp_vec{
  exp_vec(){
    set_unit_name(dut);
    set_direction(output);
    set_vc_timeout(stim_vec.get_vc_timeout());
  }
};
csl_testbench tb{
  csl_signal clk;
  dut dut;
  tb(){
  clk.set_attr(clock);
  add_logic(clock,clk,10,ns);
}
 };
```

VERILOG CODE

```
//
```

```
string vc_name.get_output_filename();
```

DESCRIPTION :

Get the name of the output file.

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

EXAMPLE :

```
//
```

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_output_filename("vector");
  }
};
csl_vector exp_vec{
  exp_vec(){
    set_unit_name(dut);
    set_direction(output);
    set_output_filename(stim_vec.get_output_filename());
  }
};
csl_testbench tb{
  csl_signal clk;
  dut dut;
  tb(){
   clk.set_attr(clock);
   add_logic(clock,clk,10,ns);
}
};
```

VERILOG CODE

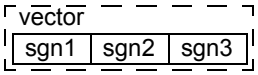
```
//
```

***vector_name*.add_signal(*signal_object*);**

DESCRIPTION :

Add the signal(s) to the vector. Signals are inserted in the vector in order from left to right. The first signal inserted into the vector is inserted in the leftmost position. The last signal inserted into the vector is inserted into the rigthmost position in the vector. The size of the vector is determined by the total width of the signals added to the vector.

FIGURE 1.2 Signals inside vector



EXAMPLE :

CSL CODE

```
//CL
csl_vector stim_vec;
//unit_a produces sgn1, sgn2, sgn3
scope unit_a{
   add_signal(output, 4, sgn1);
   add_signal(output, 4, sgn2);
   add_signal(output, 8, sgn3);
}
//add signals to the vector
stim_vec.add_signal(unit_a.sgn1);
stim_vec.add_signal(unit_a.sgn2);
stim_vec.add_signal(unit_a.sgn3);
```

VERILOG CODE

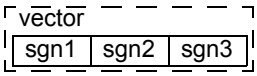
```
//CL
wire [15:0]stim_vec;
wire [3:0] sgn1, sgn2;
wire [7:0] sgn3;
//unit_a produces sgn1, sgn2, sgn3
unit unit_a(sgn1, sg2, sgn3);
//add sihnals to the vector
assign stim_vec={sgn1, sgn2, sgn3};
```

vector_name.**add_signal_group(***signal_group_name***);**

DESCRIPTION :

Add a signal group to the testbench vector. This interface can contain input signals or output signals but not both in the same time.

FIGURE 1.3 Signals inside vector



EXAMPLE :

CSL CODE

```
//CL
csl_vector stim_vec;
//unit_a produces sgn1, sgn2, sgn3
scope unit_a{
   add_signal(output, 4, sgn1);
   add_signal(output, 4, sgn2);
   add_signal(output, 8, sgn3);
}
csl_signal_group sgn;
sgn.add_signal_list(csl_list(sgn1, sgn2, sgn3);
//add signals to the vector
stim_vec.add_signal_group(sgn);
```

VERILOG CODE

```
//CL
wire [15:0]stim_vec;
wire [3:0] sgn1, sgn2;
wire [7:0] sgn3;
//unit_a produces sgn1, sgn2, sgn3
unit unit_a(sgn1, sg2, sgn3);
//add sihnals to the vector
assign stim_vec={sgn1, sgn2, sgn3};
```

```
vector_name.insert_random_pipeline_bubble();
```

DESCRIPTION :

Needs to be input vector

EXAMPLE :

//

CSL CODE

```
//
```

VERILOG CODE

```
//
```

```
vector_name.insert_random_stalls();
```

DESCRIPTION :

Needs to be output vector

EXAMPLE :

//

CSL CODE

```
//
```

VERILOG CODE

```
//
```

set_mem_instance_name(memory_instance_name);

DESCRIPTION :

Associate to a state data a memory instance. Command is mandatory if the associated unit is of other type than csl_register, csl_register_file or csl_fifo.
csl_fifo

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

EXAMPLE :

//

CSL CODE

```
csl_register_file reg_sd{
  reg_sd(){
    set_width(8);
    set_depth(256);
}
};
csl_state_data SD{
  SD(){
 set_mem_instance_name(reg_sd);
  }
};
csl_testbench tb{
  csl_signal clk_sgn;
 reg_sd reg_sd;
 tb(){
clk_sgn.set_attr(clock);
 add_logic ( clock, clk_sgn, 2 ,ns );
  }
};
```

VERILOG CODE

```
//
```

set_vc_unit_name(unit_name);

DESCRIPTION :

//

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

EXAMPLE :

//

CSL CODE

```
     csl_register_file rf(){
         set_width(4);
         set_depth(8);
       }
   };
   csl_unit top{
      csl_signal sd_clk;
      csl_signal clk_sgn;
      rf rf_;
      top(){
        sd_clk.set_attr  ( clock );
        clk_sgn.set_attr ( clock );
       }
   };
   csl_state_data tb_sd{
      tb_sd(){
       set_vc_unit_name(top);
       set_mem_instance_name (top.rf_ );
       set_vc_clock( top.sd_clk  );
       }
};
```

set_vc_clock(*signal***);**

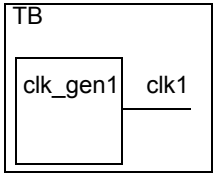
DESCRIPTION :

Sets the clock signal that triggers the capture of the vector or state data. Note that vectors and state data on the generated RTL code operate on different clock signals. Each Testbench unit have one or more clk generators. vc_clock has to be connected to one of the testbench clock signals. It is illegal to connect the vc_clock to any signal declared outside the testbench scope.

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

EXAMPLE :

FIGURE 1.4 One clock generators inside a testbench



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_vector stim_vec{
  stim_vec(){
    set_unit_name(dut);
    set_direction(input);
    set_vc_clock(clk);
      }
};
csl_vector exp_vec{
  exp_vec(){
   set_unit_name(dut);
   set_direction(output);
}
};
csl_testbench tb{
csl_signal clk(wire);
dut dut_1(.clk(clk));
tb(){
  clk.set_attr(clock);
```

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

    }
    };
```

VERILOG CODE

```
    //timescale 1ns/1ps
    module sd;
        reg clk1,clk2;
        initial
        clk1 = 0;
        always #5 clk1=~clk1;
    endmodule
```