1.1 CSL Testbench Examples

FIGURE 1.1 Use on the PLI example

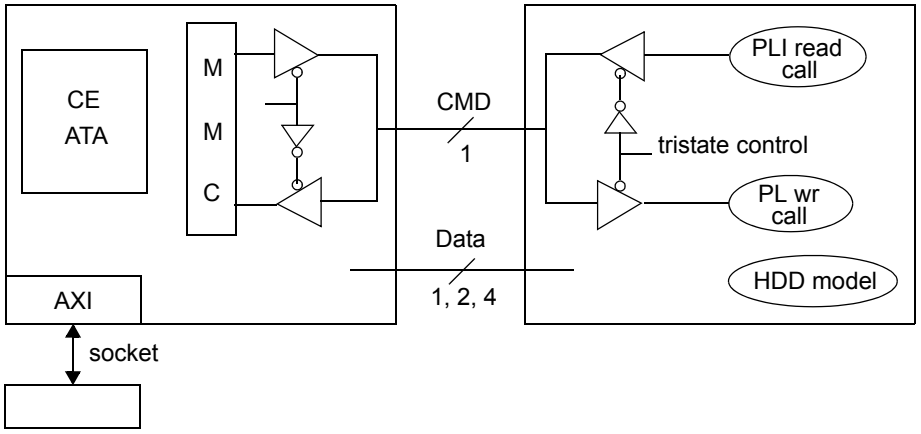
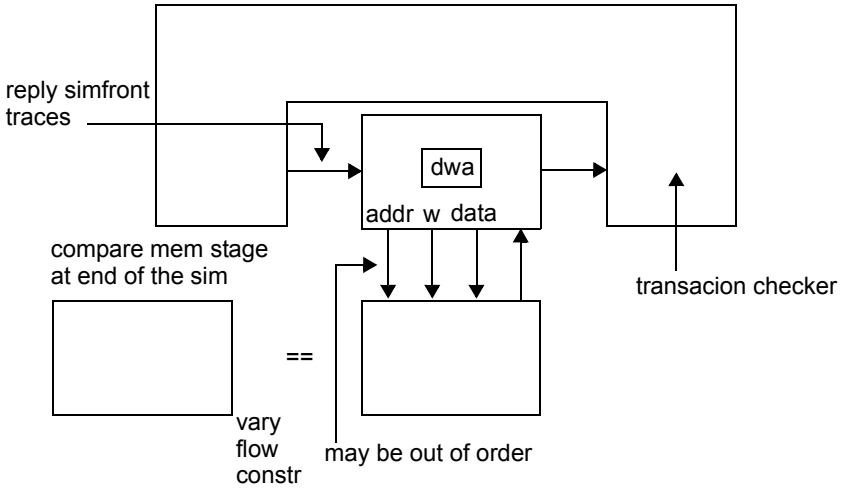


FIGURE 1.2

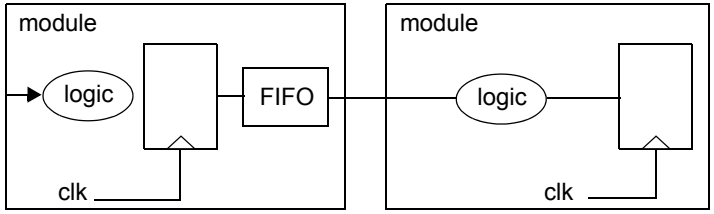


1.1.0.1 Selftest Testbench

This type of testbench doesn't need to be field with stimulus vectors and it doesn't need expected verification components.

Below there is a FIFO selftest Testbench example.

FIGURE 1.3 FIFO selftest testbench



Automatically test a FIFO

- create a tb and instantiate the fifo
- load the fifo with a sequence of numbers (0,1,2,3,4..) use a counter to create the numbers
- pop the fifo and check the sequnce for skipped numbers or complicate numbers
- vary the relationship of the clocks
- vary the order of push and pop operations
- test burst push and pop
- test single push and pop
- test single push and burst pop
- test burst push and single

Dynamically load shared objects
the C++ Simulator contains list of module names
A function will call dl_open() on each module name
A path will pe searched
if the <mn> so is not found then the dl_errors message will be printed out

1.1.1 Serial Bus (Tristate) testbench

A testbench which contains a serial bus which is used to inject and test vectors. A parallel to serial converter is used to drive transactions into a bus transaction generator. The bus transaction generator.

FIGURE 1.4 Serial bus with tristate drivers/receivers

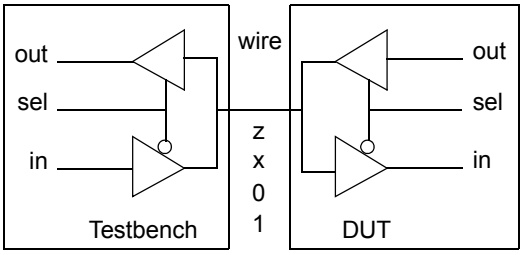
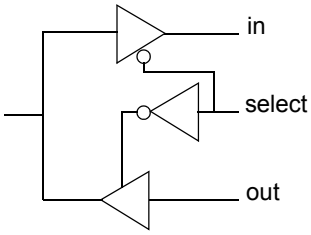


FIGURE 1.5 Testbench drives the tristate device

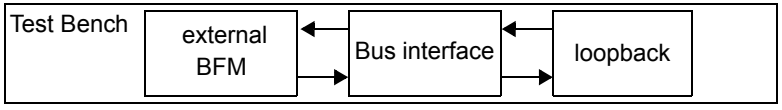


The tristate bus simulation may cause any value (i.e., x, z, 0, 1) to be driven on the bus. The serial bus testbench will assert error when an ‘x’ is driven on the serial bus.

Tristate bus conditions

- wire must not be X
- wire must be Z when in state
- wire must be 0/1 when in state
- wire must be 0/1 when data tx
- wire must be 0/1 when data rx

FIGURE 1.6 Testbench for bus interfaces



1.1.2 Testbench finish

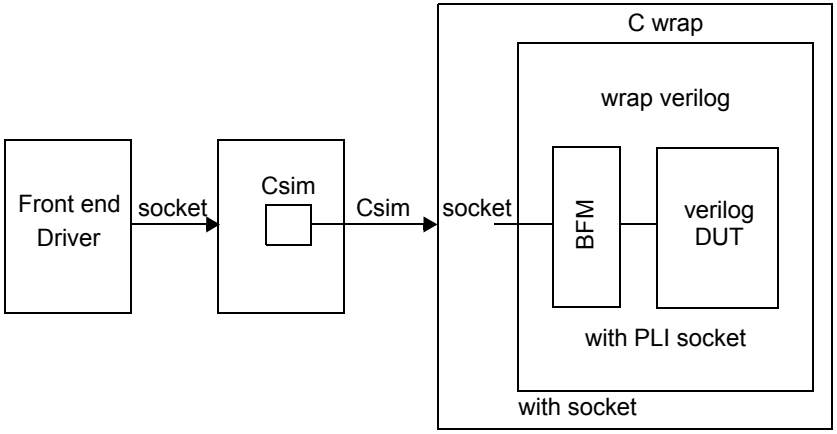
The testbench should terminate the simulation if one of the following conditions are true:
1. the correct number of output vectors has been detected
2. the time out counter has triggered
3. the mismatch counter has exceeded a match error count
4. both the input and the output vector memories have to be at their max addresses

If during normal operation the number of DUT input vectors does not equal the number of DUT output vectors then a valid bit or completion signal in the DUT is required to indicate when to compare

vectors and when the simulation is finished. If the DUT does not have a transaction complete signal then there must be a deterministic ratio between the number of DUT input vectors and the number of DUT output vectors in order to determine whenthe simulation is finished.

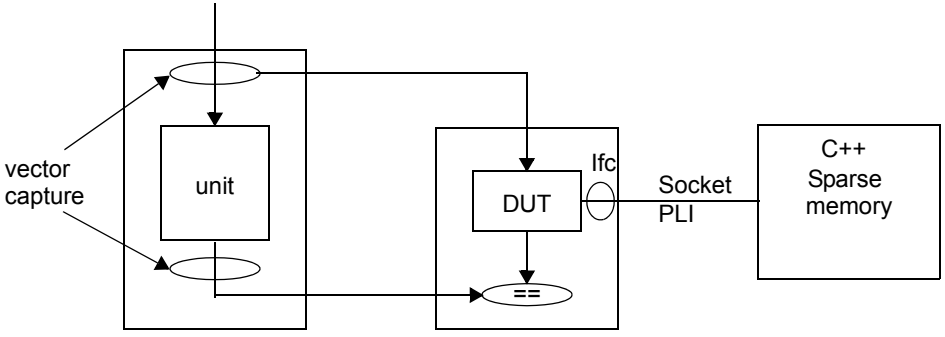
We should stop the test if the number of mismaches exceedes the number of maches because once a certain number of the expected vectors do not match the DUT output vector a bug has been found and the user needs to fix the problem. Also there are DUT's which have the following behavior: once a certain type of mismatch occurs the rest of the sequence will not match. It depends on the type of error that occurs when the test is run. If the error is an arithmetic precision error then there may be a few mismatches. If the error is a address sequencing problem or a state machine that is stuck in a loop then after the first error the rest of the vectors will probably not match.

FIGURE 1.7 Front end



The C++ Simulator must be able to drive the Verilog DUT at the system chip or unit levels

FIGURE 1.8 Socket based Testbench

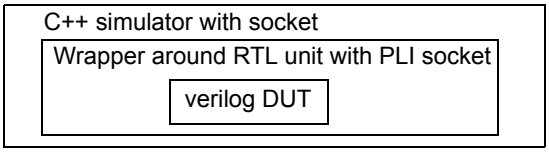


Sparse memory model - used to model large memories by only allocating memory for the parts of the moemory where a memory word is used.

1.1.3 TB interfaces

1.1.3.1 C++ Simulator and testbench communication via PLI sockets

FIGURE 1.9 C++ simulator driving RTL unit directly



Stimulus and expect vectors are generated by the C++ Simulator and used by the Veriog testbench.

FIGURE 1.10 File based communication

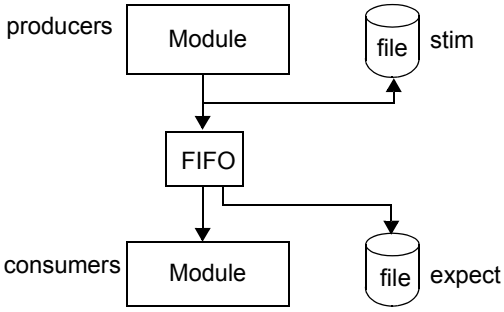
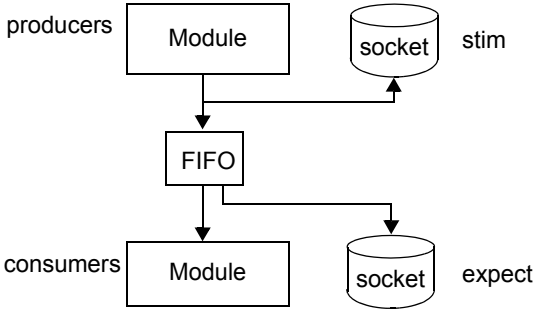


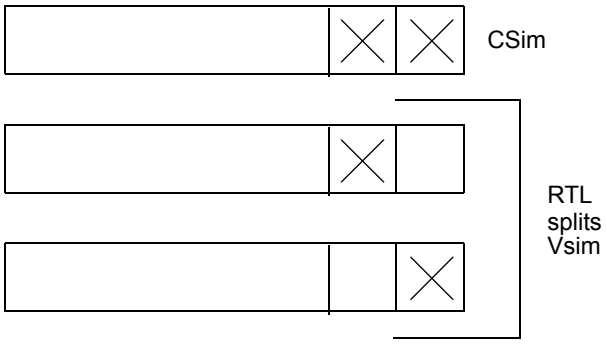
FIGURE 1.11 Socket based communication



Example csl_arch_state spec:

```
csl_arch_state as;
as.reset_name (rst_); //reset signal name
as.clk_name (clk); //clock name
as.stall_name (stall); // stall signal name
as.event_type (CLK_ENENT);
as.id('012?); // architecture state ID
as.num_mem_images //architecture state memory image
as.mem_instance_name (mem);//number of vectors to hold in tb unit
as.module_name (b);
```

FIGURE 1.12 Memory transaction ordering between C model and verilog



1.1.3.2 Model Comparation Points

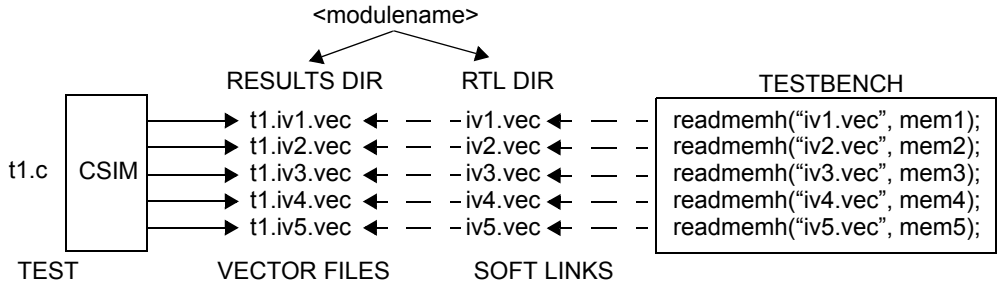
Table 1.1, “Model comparison points,” on page 6 shows the model comparison points in C++ model and RTL model.]

TABLE 1.1 Model comparison points

Simulator	System Memory	chip interface	system interface	unit vector	unit state
C++ algorithmic model	X				
C++ behavioral model	X	X	X	X	X
RTL model	X	X	X	X	X

The C++ simulator generates a vector file memory using a readmemh statement.
The connection between C++ simulator and the testbench is made using verification components.
Below is shown a vector connection between Csim and the testbench.

FIGURE 1.13 CSim write/Testbench read



1.2 Commands summary

CSL signal pattern generator -> everything below is v2.0

Common commands

```
set_timebase(ps | ns | us | ms | s);
set_generator_type(oscillator|clk|pwl);
```

Clock generator

```
set_period(numeric_expresion);
```

Signal pattern generator

```
csl_signal_pattern_generator sign_pattern_gen_name;
```

Clock div

```
    csl_clk_div clk_div_name(in_clk, out_clk, amount, [phase_diff]);
set_clk_divider(clk_signal_name);
set_clk_multiplier(clk_signal_name);
set_phase_difference(phase_diff,clk_signal_name);
set_pattern({time[ps|ns|us|ms|s], value}+);
set_start_signal(signal_object_name);
set_default_value(number);
set_reset(signal_name);
set_duration(time_range | continous);
set_signal_name(signal_object_name);
sg_output_sgn_name(signal_object_name);
sg_set_pattern(time |time_expresion, value);
```

1.3 Common commands

```
set_timebase(ps | ns | us | ms | s);
```

DESCRIPTION :

//

EXAMPLE :

//

CSL CODE

```
//
```

VERILOG CODE

```
//
```

set_generator_type(oscillator|clk|pwl);

DESCRIPTION :

oscillator - signal repeats and the signal type is wire.
clock - sets the CSL signal type to clock and uses an oscillator
pwl - a piecewise linear waveform

EXAMPLE :

//

CSL CODE

//

VERILOG CODE

//

1.3.1 Testbench clock generator

set_period(numeric_expresion);
DESCRIPTION :
//
EXAMPLE :
//
CSL CODE
//
VERILOG CODE
//

1.3.2 CSL signal generator

csl_signal_pattern_generator *sign_pattern_gen_name;*

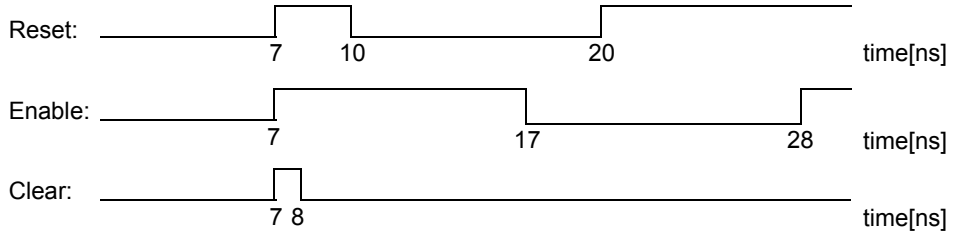
DESCRIPTION :

A csl_signal_generator is used in the testbench to generate a signal pattern. The signal pattern drives a signal which is connected to either a testbench DUT or another signal or module in the testbench.

EXAMPLE :

Using csl_signal_pattern_generator three signal are generated.

FIGURE 1.14 Waveform pattern



CSL CODE

```
//CL+AV
csl_signal_pattern_generator spgn1;
spgn1.signal_name(Reset);
spgn1.set_pwl(0ns, 0);
spgn1.set_pwl(+7ns, 1);
spgn1.set_pwl(+10ns, 0);
spgn1.set_pwl(+20ns, 1);
csl_signal_pattern_generator spgn2;
spgn2.signal_name(Enable);
spgn2.set_pwl(0s, 0);
spgn2.set_pwl(7ns, 1);
spgn2.set_pwl(0.017us, 0);
spgn2.set_pwl(0.000028ms, 1);
csl_signal_pattern_generator spgn3;
spgn3.signal_name(Clear);
t0 = 0ns;
spgn3.set_pwl(t0, 0);
t1 = t0 + 7ns;
spgn3.set_pwl(t1, 1);
t2 = t1 + 1ns;
spgn3.set_pwl(t2, 0);
```

VERILOG CODE

```
//CL
```

```
`timescale 1ns/1ps;
reg Reset, Enable, Clear;
initial begin
  Reset=0;
  Enable=0;
  Clear=0;
  #7;
  Reset=1;
  Enable=1;
  Clear=1;
  #1;
  Clear=0;
  #2;
  Reset=0;
  #7;
  Enable=0;
  #3;
  Reset=1;
  #8;
  Enable=0;
end
```

1.3.3 Clock div

```
set_clk_divider(clk_signal_name);
```

DESCRIPTION :

//

EXAMPLE :

//

CSL CODE

```
//
```

VERILOG CODE

```
//
```

```
set_clk_multiplier(clk_signal_name);
```

DESCRIPTION :

//

EXAMPLE :

//

CSL CODE

```
//
```

VERILOG CODE

```
//
```

set_phase_difference(phase_diff,clk_signal_name);

DESCRIPTION :

//

EXAMPLE :

//

CSL CODE

//

VERILOG CODE

//

set_pattern({time[ps|ns|us|ms|s], value}+);

DESCRIPTION :

The param is composed by one or more pairs. The time base is optional and if not specified it uses the general timebase or the default one.[Stefan] add the stuff here

EXAMPLE :

//

CSL CODE

//

VERILOG CODE

//

set_start_signal(signal_object_name);

DESCRIPTION :

This tells the pattern generator to start generating the signal. The start signal activates the pattern generator.

EXAMPLE :

//

CSL CODE

//

VERILOG CODE

//

set_default_value(number);

DESCRIPTION :

The default value is the value driven by the pattern generator when it is inactive. for example, a reset signal may be low true so the default value should be high (1).

EXAMPLE :

//

CSL CODE

//

VERILOG CODE

//

```
set_reset(signal_name);
```

DESCRIPTION :

The default value is the value driven by the pattern generator when it is inactive. for example, a reset signal may be low true so the default value should be high (1).

EXAMPLE :

//

CSL CODE

```
//
```

VERILOG CODE

```
//
```

```
set_duration(time_range | continous);
```

DESCRIPTION :

//

EXAMPLE :

//

CSL CODE

```
//
```

VERILOG CODE

```
//
```

```
set_signal_name(signal_object_name);
```

DESCRIPTION :

Sets the name of the signal that is driven by the pattern generator.

EXAMPLE :

//

CSL CODE

```
//
```

VERILOG CODE

```
//
```

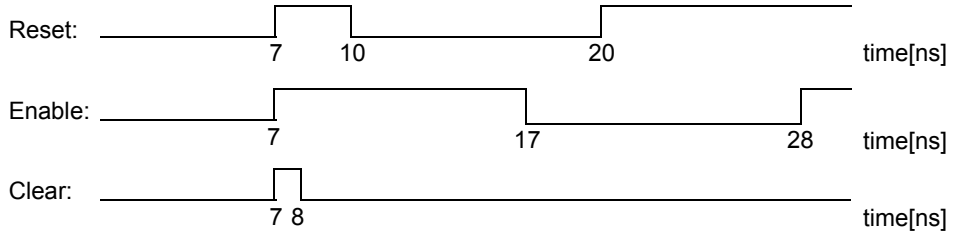
sg_output_sgn_name(*signal_object_name***);**

DESCRIPTION :

Generates a signal following a pattern. *signal_object_name* is created if this signal doesn't exist.

EXAMPLE :

FIGURE 1.15 Waveform pattern



Using csl_signal_pattern_generator three signal are generated.

CSL CODE

```
//CL+AV
csl_signal_pattern_generator spgn1;
spgn1.signal_name(Reset);
spgn1.set_pwl(0ns, 0);
spgn1.set_pwl(+7, 1);
spgn1.set_pwl(+10, 0);
spgn1.set_pwl(+20, 1);
csl_signal_pattern_generator spgn2;
spgn2.signal_name(Enable);
spgn2.set_pwl(0s, 0);
spgn2.set_pwl(7ns, 1);
spgn2.set_pwl(0.017us, 0);
spgn2.set_pwl(0.000028ms, 1);
csl_signal_pattern_generator spgn3;
spgn3.signal_name(Clear);
t0 = 0ns;
spgn3.set_pwl(t0, 0);
t1 = t0 + 7ns;
spgn3.set_pwl(t1, 1);
t2 = t1 + 1ns;
spgn3.set_pwl(t2, 0);
```

VERILOG CODE

```
//CL
`timescale 1ns/1ps;
```

```
reg Reset, Enable, Clear;
initial begin
  Reset=0;
  Enable=0;
  Clear=0;
  #7;
  Reset=1;
  Enable=1;
  Clear=1;
  #1;
  Clear=0;
  #2;
  Reset=0;
  #7;
  Enable=0;
  #3;
  Reset=1;
  #8;
  Enable=0;
end
```

sg_set_pattern(*time |time_expresion, value***);**

DESCRIPTION :

Generates a signal following a pattern. The time values can be relative or absolute.

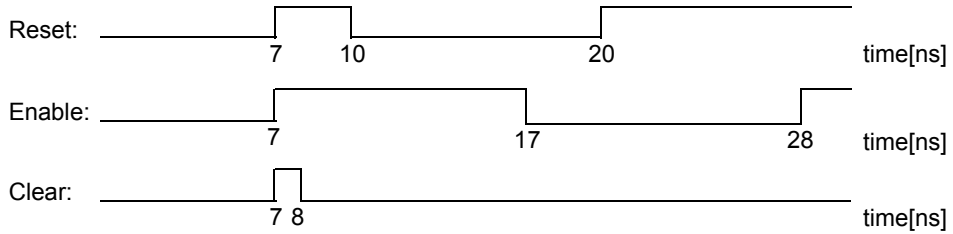
TABLE 1.2 Time units

Time unit	Meaning	10's exponent
ps	pico second	-12
ns	nano second	-9
us	micro second	-6
ms	mili second	-3
s	second	1

EXAMPLE :

Using csl_signal_pattern_generator three signal are generated.

FIGURE 1.16 Waveform pattern



CSL CODE

```
//CL+AV
csl_signal_pattern_generator spgn1;
spgn1.signal_name(Reset);
spgn1.set_pwl(0ns, 0);
spgn1.set_pwl(+7ns, 1);
spgn1.set_pwl(+10ns, 0);
spgn1.set_pwl(+20ns, 1);
csl_signal_pattern_generator spgn2;
spgn2.signal_name(Enable);
spgn2.set_pwl(0s, 0);
spgn2.set_pwl(7ns, 1);
spgn2.set_pwl(0.017us, 0);
spgn2.set_pwl(0.000028ms, 1);
csl_signal_pattern_generator spgn3;
spgn3.signal_name(Clear);
t0 = 0ns;
```

```
spgn3.set_pwl(t0, 0);
t1 = t0 + 7ns;
spgn3.set_pwl(t1, 1);
t2 = t1 + 1ns;
spgn3.set_pwl(t2, 0);
```

VERILOG CODE

```
//CL
`timescale 1ns/1ps;
reg Reset, Enable, Clear;
initial begin
  Reset=0;
  Enable=0;
  Clear=0;
  #7;
  Reset=1;
  Enable=1;
  Clear=1;
  #1;
  Clear=0;
  #2;
  Reset=0;
  #7;
  Enable=0;
  #3;
  Reset=1;
  #8;
  Enable=0;
end
```