Joint ICTP-IAEA School on Systems-on-Chip based on FPGA for Scientific Instrumentation and Reconfigurable Computing

Verification - Test Bench

Cristian Sisterna

UNSJ

A brief chronology of events:

July 1994—A summer intern discovers a bug in the floating point unit during testing; however, Intel did not expect it to be major problem for its users. Hence they continued to produce the flawed chip, while planning to produce the corrected chip starting 1995.

September 1994—A mathematics professor in Virginia, Thomas Nicely discovers the bug that causes errors in floating-point divisions with more than 5 significant digits. He reported it to Intel but got no official response from Intel.

November 1994—The EE times reported the story, but Intel claimed that the error occurs very infrequently and will not be seen by the average user. The story was picked up many newspapers including the New York Times, the San Jose Mercury News, and the San Francisco Chronicle.

December 1994—IBM stops shipping IBM PCs that used the affected Pentium chip in early December. Following that, on December 21, Intel officially apologized and announced that users could get their Pentium P5 chips exchanged for an updated processor in which the flaw is corrected.

January 1995—Intel spent $475 million against earnings to replace the flawed processors.

The bug was caused because of incorrect elements returned by the PLA lookup table used in the floating-point division in the chip. When discovered in July 1994, the cost to fix the bug was estimated to be several hundred thousand dollars, but it would take a few months to make the change, verify the chip, and produce the corrected chips.

The field of "Verification" got a lot of attention following this event, since companies are interested in avoiding this kind of economic damage and embarrassment. In the years following this event, PhD graduates with dissertations in verification were heavily sought after by chip design companies.

The Intel Pentium P5 chips with the FDIV bug looked like this.

Courtesy of Intel Corporation and CPU Collection Konstantin Lanzet

Verification

It's an IMPORTANT part of the design process:

- How do I know that the design works as expected?

- It's IMPORTANT/NECESSARY to verify the behavior of the daesign in **at least one** of following parts of the design process:
  - Functional Verification / Pre-synthesis Verification
  - Post-synthesis
  - Timing Verification / Post-Place & Route Verification

Verification, Where in the design flow ?

- *Functional Verification / Pre-synthesis Verification:*
  - Where most errors *can and should be found*
  - ALWAYS do this verification

- *Post-synthesis Verification*
  - not necessary unless several *synthesis attributes* were used and it's necessary to verify the behavior after synthesis

- *Timing Verification / Post-Place & Route Verification*
  - Necessary in high frequency designs, in the order of >100MHz
  - Time consuming verification process (very long verification time)

What is a Test Bench ?

- A VHDL model which generates stimulus waveforms to test and verify the functionality a digital system described in VHDL

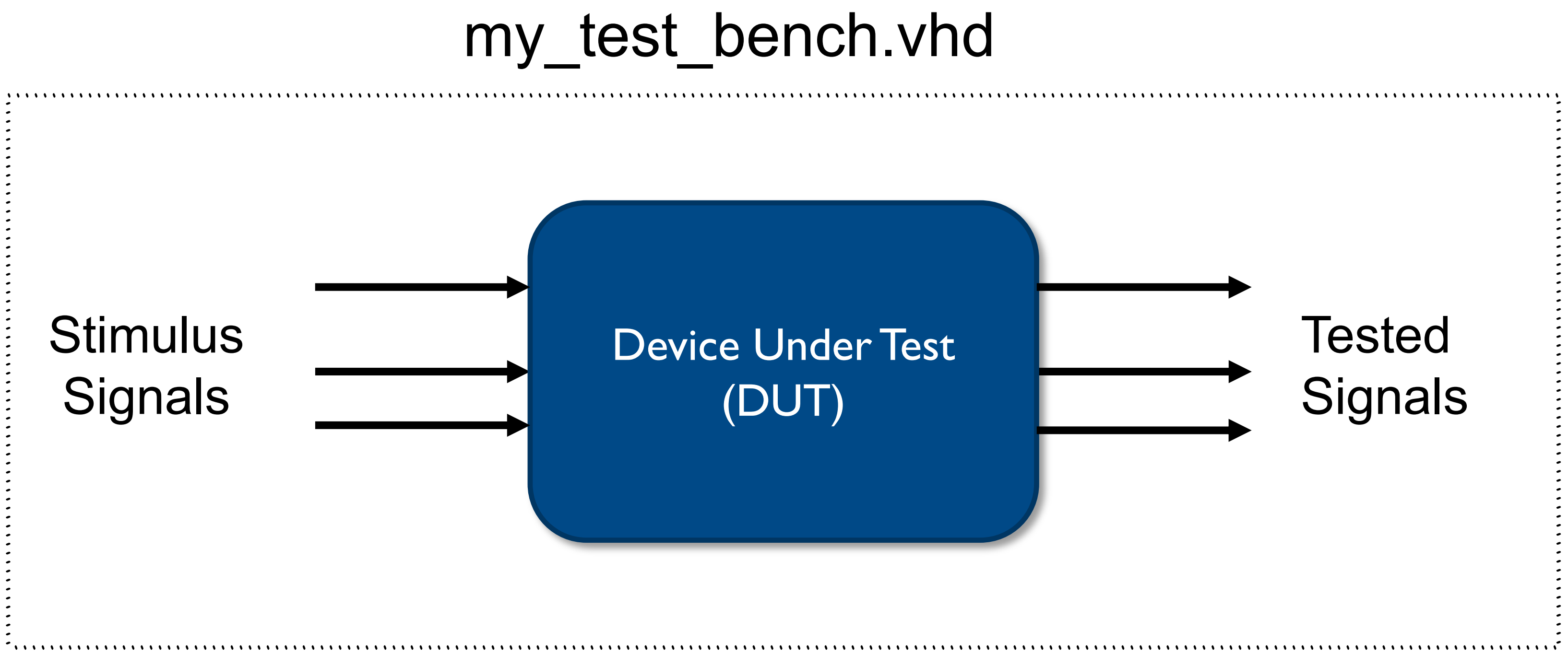
- It has three main purposes:
  - To generate a pattern stimulus for executing a simulation
  - To apply the generated stimulus to the entity under test and to collect output's responses
  - To compare output's responses with expected values

What is a Test Bench ?

- A test bench is usually executed in a simulation tool (ModelSim, ISIM, etc) where the test pattern and the outputs can seen graphically as waveforms with precise time information.

- Test bench should be created by a DIFFERENT engineer than the one who created the device under test (DUT)

Test Bench

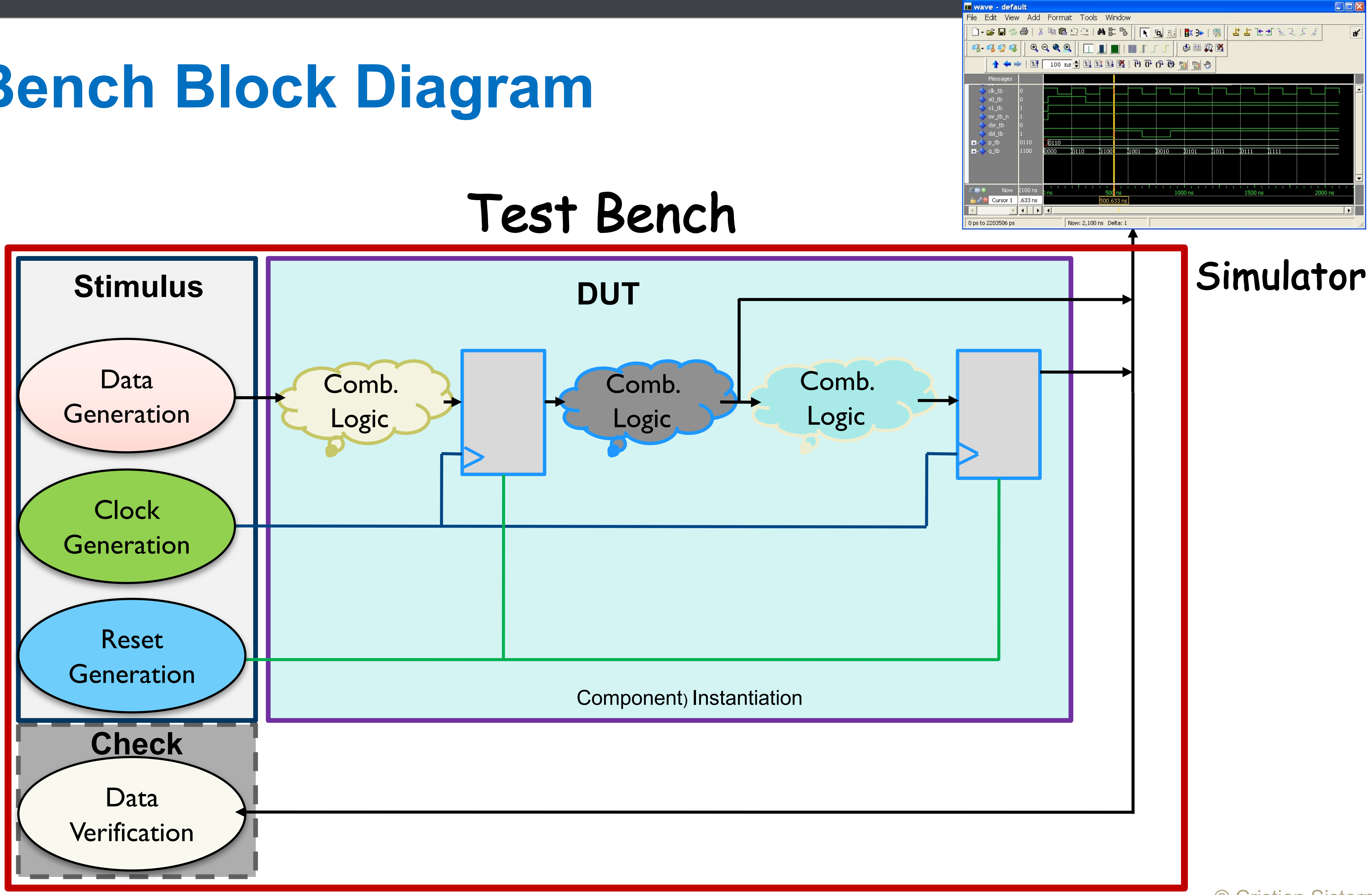

my_test_bench.vhd
Stimulus Signals
Device Under Test (DUT)
Tested Signals

VHDL Components of a Test Bench

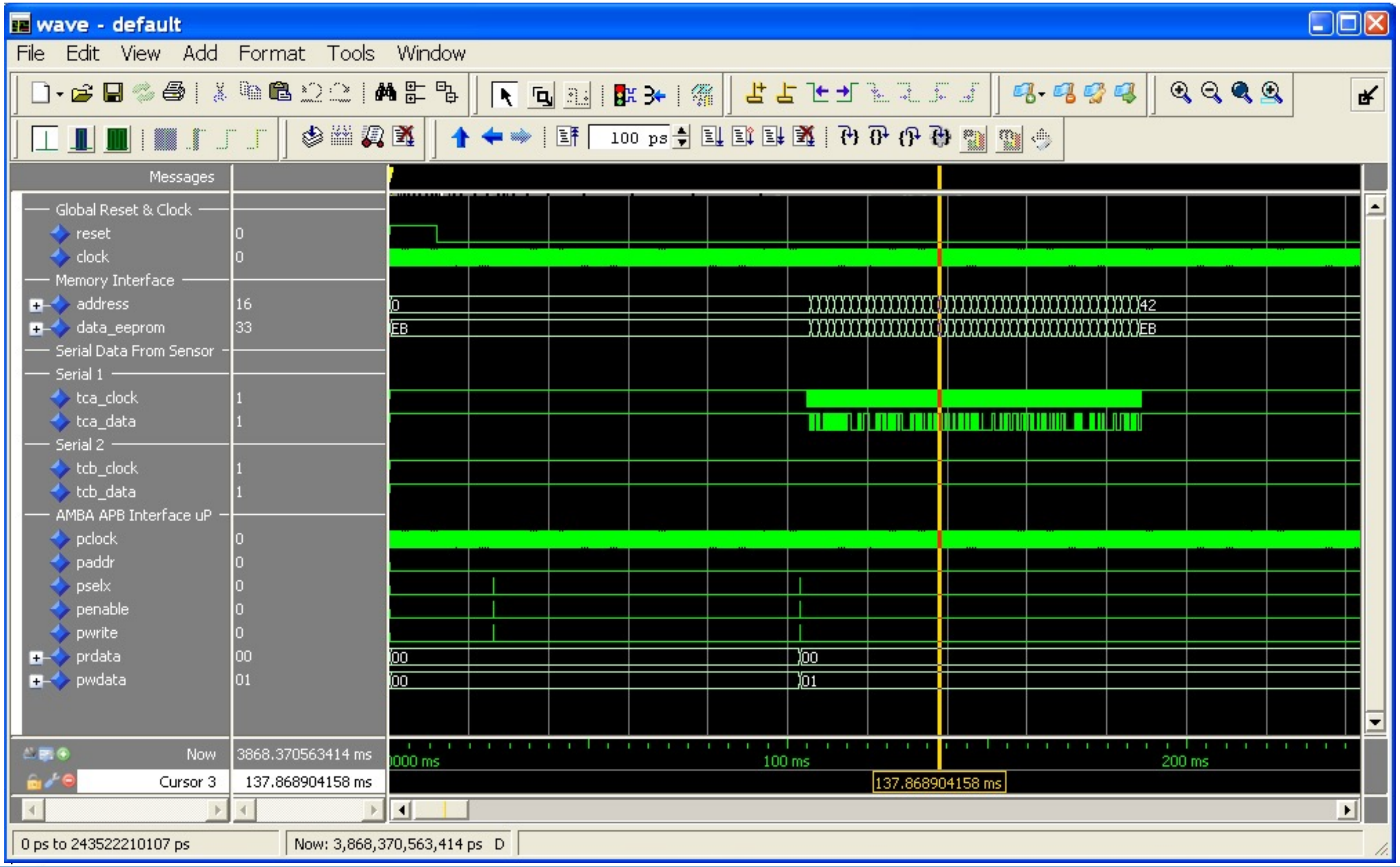
- Test bench **entity**:
  - Empty declaration

- Test bench **architecture**:
  - Component declaration
  - Local signal declaration
  - Component instantiation
  - Data stimulus generation statements
  - Clock/Reset generation statements
  - Output's values check statements (optional)

Test Bench Block Diagram



Test Bench Simulation Result - Waveforms



```
Global Reset & Clock
  reset              0
  clock              0
Memory Interface
  address            16
  data_eeprom        33
Serial Data From Sensor
Serial 1
  tca_clock          1
  tca_data           1
Serial 2
  tcb_clock          1
  tcb_data           1
AMBA APB Interface uP
  pclock             0
  paddr              0
  pselx              0
  penable            0
  pwrite             0
  prdata             00
  pwdata             01

Now        3868.370563414 ms
Cursor 3   137.868904158 ms

0000 ms   100 ms   200 ms
137.868904158 ms

0 ps to 243522210107 ps    Now: 3,868,370,563,414 ps  D
```

Test Bench VHDL Template

```
library ieee;
use ieee.std_logic_1164.all;

-- TB entity declaration del TB (empty entity)
entity testbench is
end testbench;

-- architecture declaration
architecture tb of testbench is
-- component declaration: component to test
        component device_under_test
                port(list_of_ports);
        end component;

-- local signal declarations. Used to:
-- stimulate the DUT's inputs
-- test the DUT's outputs
        <local_signal_declarations;>
```

Test Bench VHDL Template

```
begin
-- component instantiation:
-- associate the top-level (TB)
-- signals to their equivalent DUT’s signals

  DUT: entity_under_test port map( list_of_ports);

-- stimulus statements for the input signals.
-- values are assigned at different times per each input

    generate_input_waveforms;

-- ouptut signals’ check

    monitor_output_statements; -- optional

end tb;
```

Test Bench Template - Example

```
library ieee;
use ieee.std_logic_1164.all;

entity test_my_design is
end test_my_design;

architecture testbench of test_my_design is
  component my_design is
     port (a,b : in std_logic;
           x,y : out std_logic);
  end component;

  signal as,bs : std_logic:='1';
  signal xs,ys : std_logic;

begin
  uut : my_design port map
        (a=>as, b=>bs, x=>xs, y=>ys);

    as <= not(as) after 50 us;

  process begin
    bs <= '1'; wait for 75 us;
    bs <= '0'; wait for 25 us;
  end process;
end testbench;
```

} Define library, same as in VHDL source code

} VHDL model without entity interface

} Component declaration of the device to test

} Define signal names

} Instantiated UUT in test bench

} Define the stimulus for the inputs of the component under test

Most Common VHDL Statements for Test Bench

wait statement

- The wait statement can be located anywhere between **begin** and **end** process

- Basic Usages:

```
wait for time;
 wait until condition;
wait on signal_list;
wait;
```

Use of *wait* (1)

```
process
   . . .
  J <= '1';
  wait for 50 us;    --  process is suspended for 50 ns after J is
 . . .                -- assigned to '1'
end process;
```

```
process
   . . .
  wait until CLK = '1';-- sync with CLK rising edge before
                       -- continuing of simulation
  . . .
end process;
```

Use of *wait* (2)

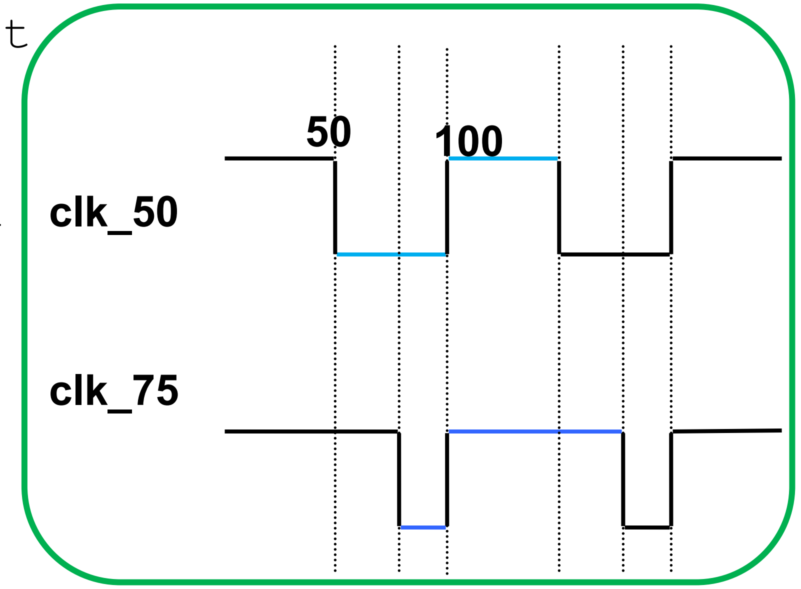
```
    process
      . . .
      wait on A until CLK = '1';-- the process is resumed
                                -- after a change on A signal,
                                -- but only when the value of
                                -- the signal CLK is equal to '

      . . .
    end process;
```

```
    process
        rst <= '1';
        wait for 444 us;
        rst <= '0';
        wait;        -- used without any condition,
                     -- the process will be suspended
                     -- forever
    end process;
```

Data Stimulus Generation

Periodic Signal Generation

```
architecture testbench of test_my_design is
  signal clk_50 : std_logic := '1';
  signal clk_75 : std_logic := '0';
  constant clk_period: time := 100 us;
  constant h_clk_period: time := 50 us;

begin
 -- case 1: concurrent statement
 clk_50 <= not(clk_50) after h_clk_period;-- 50% duty

 -- case 2: sequential statement
 clk_75_proc: process
 begin
    clk_75 <= '1';
    wait for 75 us; -- 75% duty
    clk_75 <= '0';
    wait for 25 us;
 end process clk_75_proc;
 . . .
 . . .
end testbench;
```



Stimulus Generation

- Avoid race conditions between data and clock
  - Applying the data and the active edge of the clock simultaneously might cause a race condition
  - To keep data synchronized with the clock while avoiding race condition, apply the data at a different point in the clock period that at the active edge of clock

Data Generation (1)

Example of Data generation on inactive clock edge

Clock generation process

```
clk_gen_proc: process
begin
 clk <= '0';
 wait for 25 ns;
 clk<= '1';
 wait for 25 ns;
end process clk_gen_proc;
```

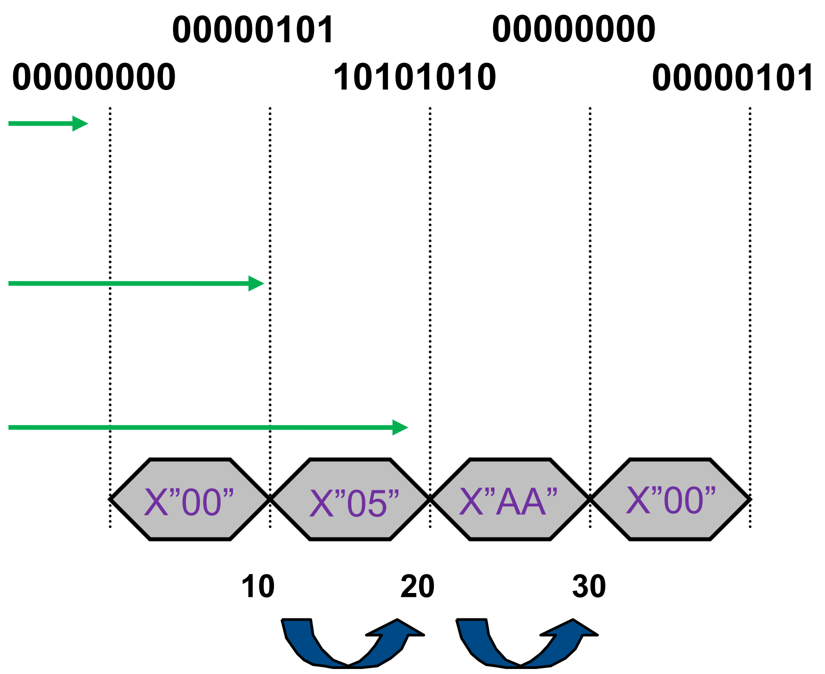
Data generation process

```
data_gen_proc: process
while not (data_done) loop
   DATA1 <= X1;
   DATA2 <= X2;
   ...
   wait until falling_edge(clk);
 end loop;
end process data_gen_proc;
```

Data Generation (2)

Relative time: signal waveforms that are specified to change at simulation times relative to the previous time, in a time accumulated manner

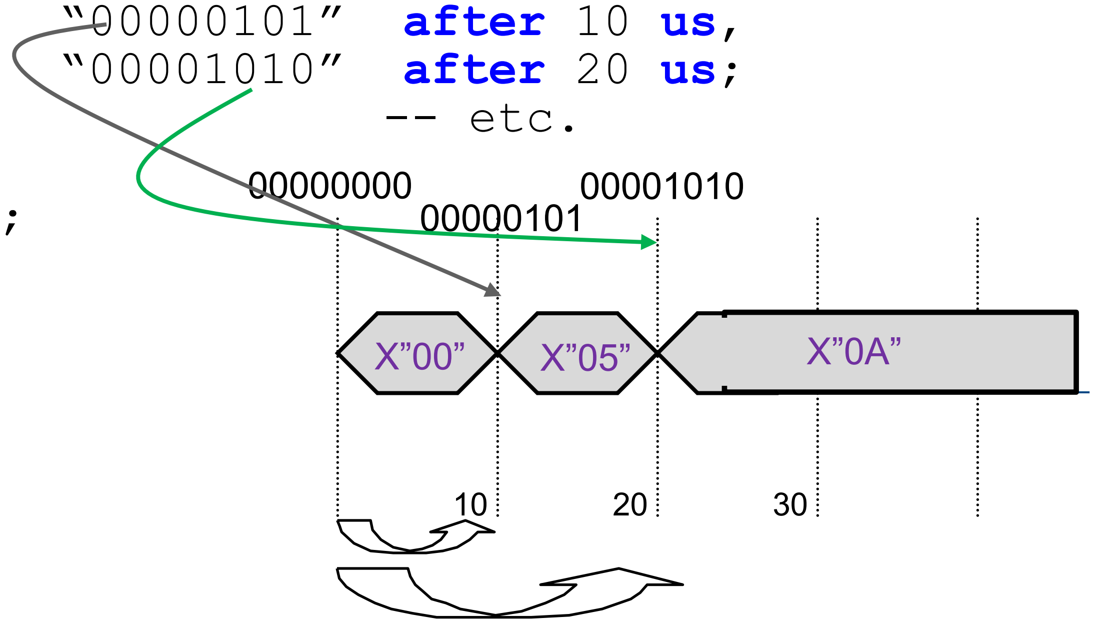
```
architecture relative_timing of myTest is
     signal Add_Bus : std_logic_vector(7 downto 0);
begin
   patt_gen_proc: process
   begin
        Add_Bus <= "00000000";
        wait for 10 us;

        Add_Bus <= "00000101";
        wait for 10 us;

        Add_Bus <= "10101010";
        wait for 10 us;
   end process patt_gen_proc;
. . .
end relative_timing;
```



Data Generation (3)

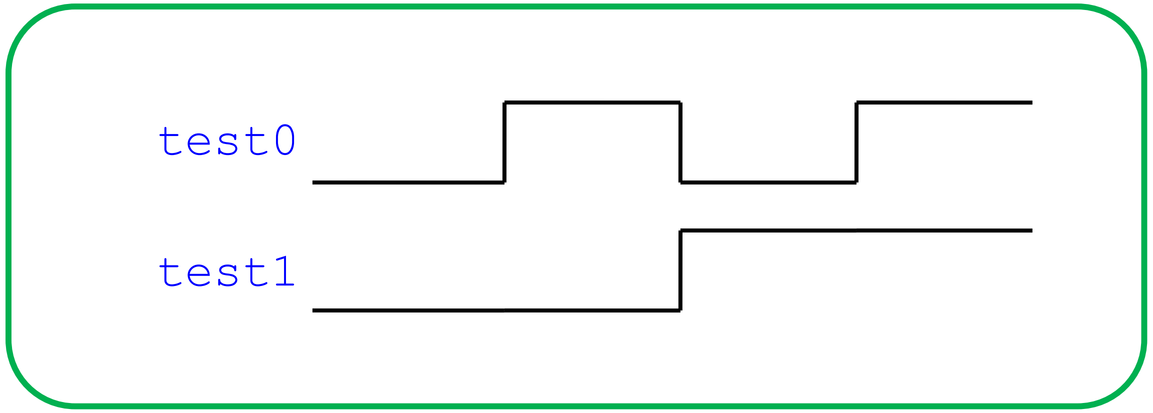
Absolute time: signal waveforms that are specified to change at simulation times absolute since the moment that the simulation begin

```
architecture absolute_timing of testbench is
     signal A_BUS : std_logic_vector(7 downto 0);

begin
        A_BUS<= "00000000",
                          "00000101"  after 10 us,
                          "00001010"  after 20 us;
                                   -- etc.
. . .
end absolute_timing;
```



Data Generation (4)

```
-- 2 bits test pattern
begin
       test0 <= '0', '1' after 10 us, '0' after 20 us,
                   '1' after 30 us;

       test1 <= '0',
                  '1' after 20 us;
. . .
```



Data Generation (5)

```
architecture array_usage of in_test_benches is
   signal add_bus : std_logic_vector(7 downto 0);
   -- type & signal declarations: 5 data for 8 bits
   subtype stimulus is array (0 to 4) of
                                  std_logic_vector (7 downto 0);

   constant data : stimulus :=
             ("00000000",          -- declare the stimulus
              "00000001",          -- as an array.
              "00000010",          -- these values will be
              "00000011",          -- used to stimulate the
              "00000100");         -- inputs
begin
   stim_proc: process
   begin
        for i in 0 to 4 loop             -- for loop that
assign
           add_bus <= data(i); -- to add_bus a new value
           wait for 10 us;     -- from stimulus every 10ns
        end loop;
   end process stim_proc;
. . .
end array_usage;
```

Data Generation (6)

```
architecture array_usage of in_test_benches is

-- same declarations as previous example

begin
  process
  begin
       for i in 0 to 4 loop
            Add_BUS <= DATA(i);
              wait until falling_edge(clk);
             for k in 1 to 7 loop
                   wait until rising_edge(clk);
             end loop;
       end loop;
  end process;
. . .
end array_usage;
```

In this case each pattern in the sequence is held for how many clock cycles???

Reset Generation

```
-- asynchronous desassert
   reset
reset: process
begin
  rst <= ’1’;
  wait for 23 us;
  rst <= ’0’;
  wait for 1402 us;
  rst <= ’1’;
  wait for 23 us;
  rst <= ’0’;
  wait;
end process;
```

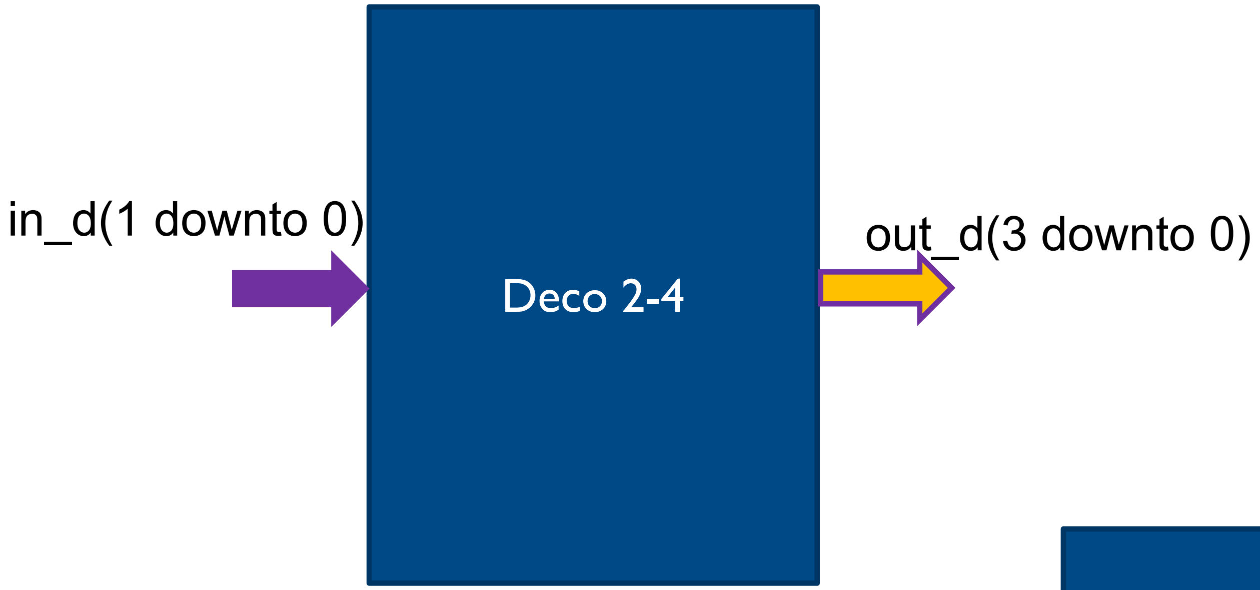
```
-- synchronous desassert reset
sreset: process
begin
  rst <= ’1’;
  for i in 1 to 5 loop
    wait until clk = ‘1’;
  end loop;
  rst <= ’0’;
  wait;
end process;
```

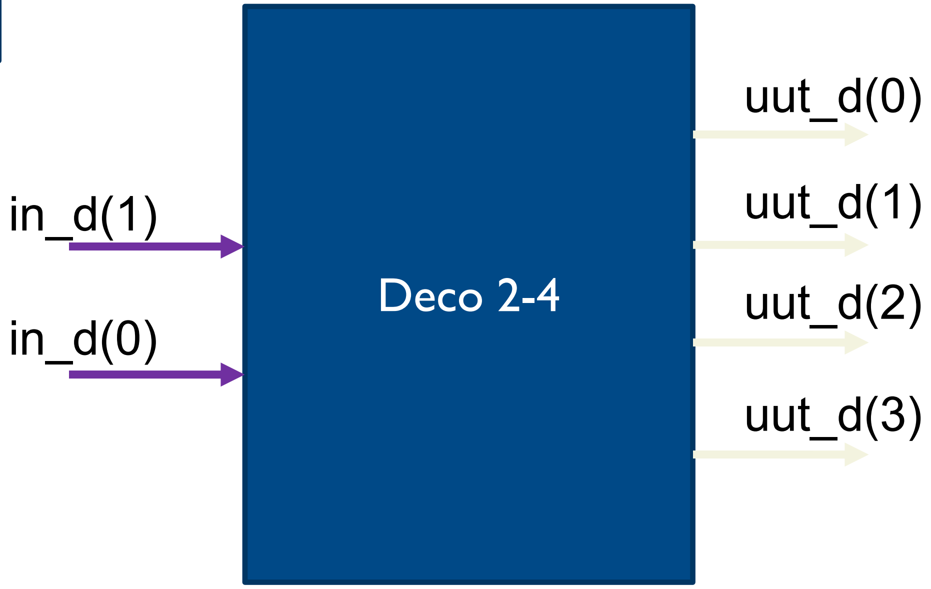
Simple Example

2:4 Decoder

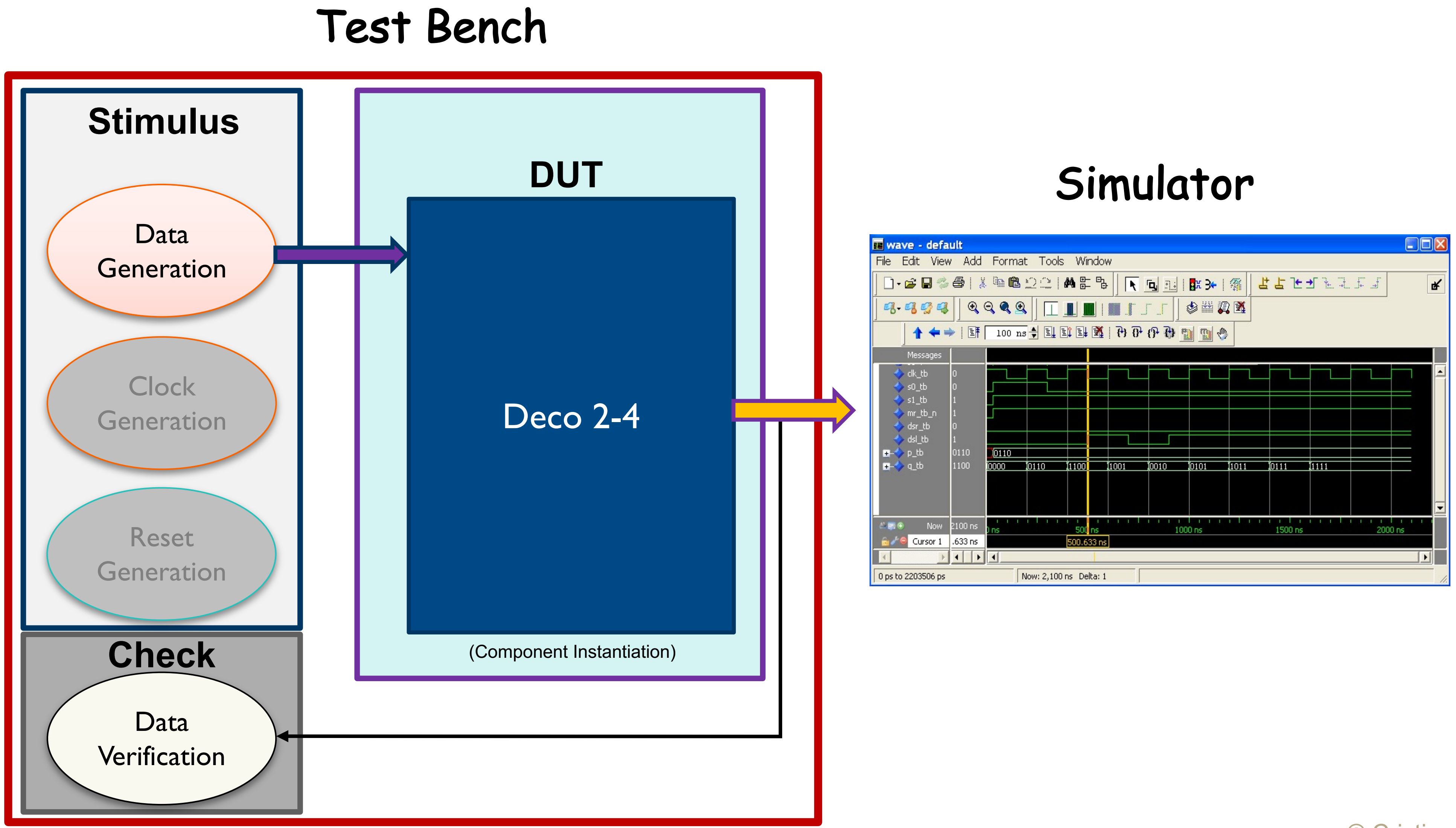
```
library Library ieee;
use ieee.std_logic_1164.all;

entity decoder_2_4 is
  port (in_d  : in  std_logic_vector (1 downto 0);
        out_d : out std_logic_vector (3 downto 0));
end decoder_2_4 ;
--
architecture dataflow of decoder_2_4 is
begin
  with in1 select
    out1 <= "0001" when "00",
            "0010" when "01",
            "0100" when "10",
            "1000" when "11",
            "0000" when others;
end dataflow;
```

How to test ‘Simple Decoder”??





Test Bench for Simple 2:4 Decoder



Test Bench 1

```
library ieee;
use ieee.std_logic_1164.all;

-- Test Bench to exercise and verify
-- correctness of decoder_2_4 entity
entity tb_case1_decode is
end tb_case1_decode ;

architecture tb_c1 of tb_case1_decode is
-- signal declarations
 signal in_tb  : std_logic_vector (1 downto 0);
 signal out_tb : std_logic_vector (3 downto 0);
-- component declaration
component decode
      port (
             in_d : in  std_logic_vector(1 downto 0);
             out_d: out std_logic_vector(3 downto 0));
end component;
```

Test Bench 1

```
begin
 dut: decode port map(
                        in_d  => in_tb,
                        out_d => out_tb);

 patt_gen_proc: process
   begin
   in_tb<= “00”;
   wait for 10 us;

   in_tb <= “01”;
   wait for 10 us;

   in_tb <= “10”;
   wait for 10 us;

   in_tb <= “11”;
   wait for 10 us;

   end process patt_gen_proc;
 . . .
 end architecture tb_c1 ;
```

Component Instantiation- Inputs Stimulus and Outputs port map

Stimulus generation

Test Bench 2

```
library ieee;
use ieee.std_logic_1164.all;

-- Test Bench to exercise and verify
-- correctness of decoder_2_4 entity
entity tb_case2_decode is
end tb_case2_decode ;

architecture tb_c2 of tb_case2_decode is
-- signal declarations
 signal in_tb  : std_logic_vector (1 downto 0);
 signal out_tb : std_logic_vector (3 downto 0);
-- component declaration
component decode
      port (
             in_d : in  std_logic_vector(1 downto 0);
             out_d: out std_logic_vector(3 downto 0));
end component;
```

Test Bench 2

```
begin
 dut: decode port map(
                         in_d  => in_tb,
                         out_d => out_tb);



 begin
  in_tb <= “00”,
          “01”  after 10 us,
              “10”  after 20 us,
              “11”  after 20 us;

 . . .
 end architecture tb_c2;
```

Component Instantiation-Inputs Stimulus and Outputs port map

Stimulus generation

Test Bench 3

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

-- Test Bench to exercise and verify
-- correctness of decoder_2_4 entity
entity tb_case3_decode is
end tb_case3_decode ;

architecture tb_c3 of tb_case3_decode is
-- signal declarations
 signal in_tb  : std_logic_vector (1 downto 0);
 signal out_tb : std_logic_vector (3 downto 0);
-- component declaration
component decode
      port (
             in_d : in  std_logic_vector(1 downto 0);
             out_d: out std_logic_vector(3 downto 0));
end component;
```

Test Bench 3

```
signal count: unsigned(1 downto 0);--numeric_std
begin
dut: decode port map(

   in_d  => in_tb,
                                                        out_d
   => out_tb);

apply_inputs: process
begin
   wait on (rising_edge(clk));
         count <= count + 1;
end process apply_inputs;

 in_tb <= std_logic_vector(count);

end architecture tb_c3;
```

Component Instantiation- Inputs Stimulus and Outputs port map

Data generation

Any error?

Test Bench 4

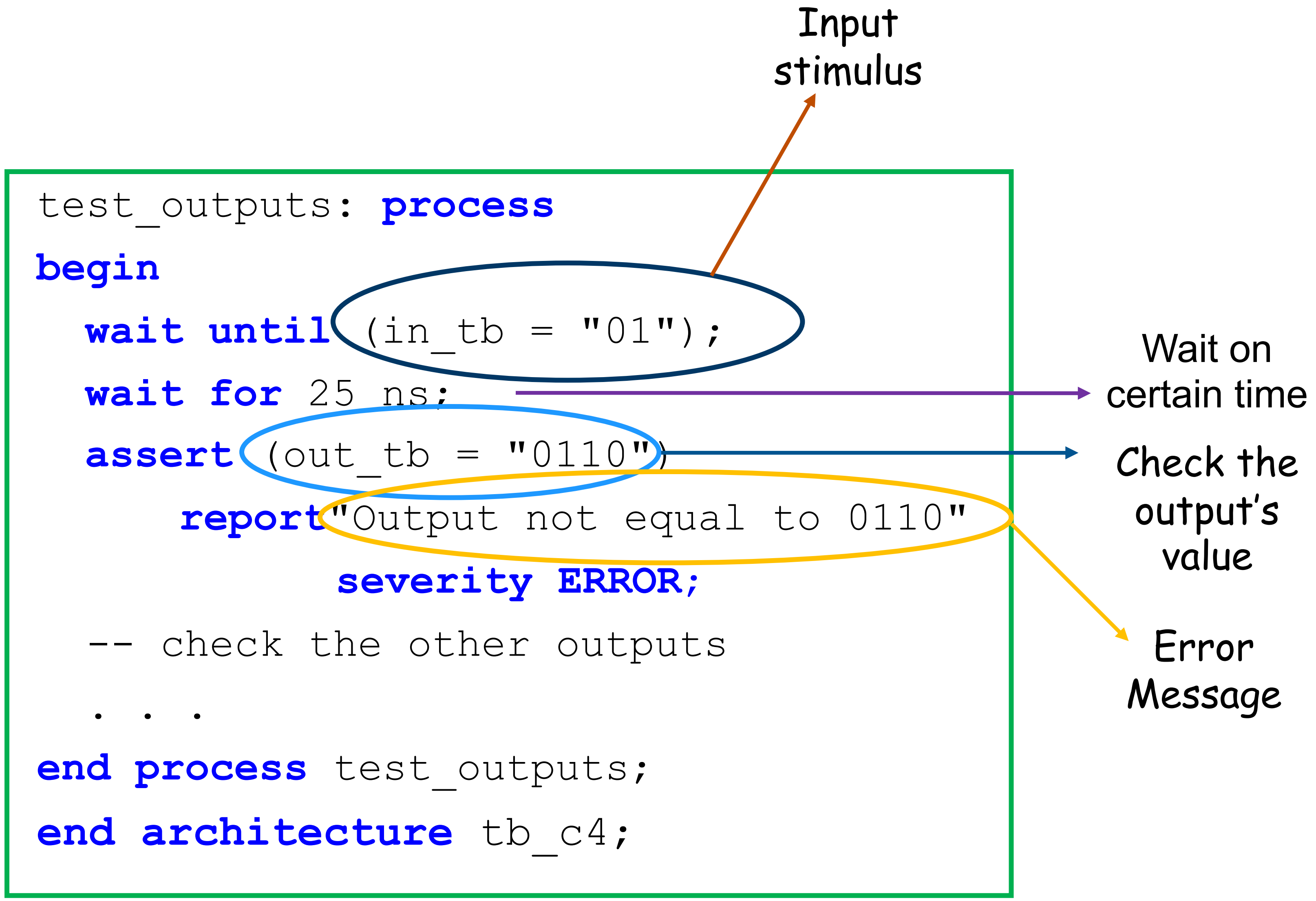
```
library ieee;
use ieee.std_logic_1164.all;

entity tb_case4_decode is
end tb_case4_decode;

architecture tb_c4 of tb_case4_decode is

subtype input_array is array(0 to 3) of
                                   std_logic_vector(1 downto 0);
constant input_vectors: input_array :=
                                   ("00", "01", "10", "11");
signal in_tb  : std_logic_vector (1 downto 0);
signal out_tb : std_logic_vector (3 downto 0);

component decode
      port (
             in_d : in  std_logic_vector(1 downto 0);
             out_d: out std_logic_vector(3 downto 0));
end component;
```

Test Bench 4

```
begin
dut: decode port map(
                        in_d  => in_tb,
                        out_d => out_tb);

apply_inputs: process
begin
   for j in input_vectors'range loop
        in_tb <= input_vectors(j);
        wait for 50 ns;
   end loop;
   wait; -- ???
end process apply_inputs;

-- verification process
```

Data generation

Test Bench 4



```vhdl
test_outputs: process
begin
  wait until (in_tb = "01");
  wait for 25 ns;
  assert (out_tb = "0110")
      report"Output not equal to 0110"
            severity ERROR;
  -- check the other outputs
  . . .
end process test_outputs;
end architecture tb_c4;
```

Test Bench 5

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity tb_case5_decode is
end tb_case5_decode;

architecture tb_c5 of tb_case5_decode is
type decoder_test is record
  in_tb_stimulus: std_logic_vector(1 downto 0);
  out_tb_stimulus: std_logic_vector(3 downto 0);
end record;
type test_array is array(natural range <>) of decoder_test;
constant test_data: test_array :=
   (("00", "0001"),
    ("01", "0010"),
    ("10", "0100"),
    ("11", "1000"));
-- same component declaration as before
signal in1_tb : std_logic_vector(1 downto 0);
signal out1_tb: std_logic_vector(3 downto 0);
```

Test Bench 5

```
begin
dut: decode port map(
                         in1  => in1_tb,
                         out1 => out1_tb);
 apply_in_check_outs: process
 begin
  for j in test_data'range loop
       in1_tb <= test_data(j).in_tb_stimulus);
       wait for 50 ns;
        assert (out1_tb = test_data(j).out_tb_stimulus)
            report "Output not equal to the expected value,
                   error en indice " & integer'image(j);
                severity ERROR;
  end loop;
 end process apply_in_check_outs;
end architecture tb_c5;
```

Test Bench 5

```
begin
apply_inputs: process
begin
   for j in test_data'range loop
        in1_tb <= test_data(j). in_tb_stimulus);
        wait for 50 ns;
   end loop;
end process apply_inputs;
data_verif: process
begin
  wait for 25 ns;
  assert (out1_tb = test_data(j).out_tb_stimulus)
           report "Output not equal to the expected value"
               severity ERROR;
  wait for 50 ns;
end process data_verif;
. . .
```

Synchronous 2:4 Decoder

```
entity synch_decoder_2_4 is
  port (in_d : in  std_logic_vector(1 downto 0);
        clk : in  std_logic ;
        out_d : out std_logic_vector(3 downto 0));
end synch_decoder_2_4 ;
architecture dataflow of synch_decoder_2_4 is
 signal out1_i: std_logic_vector (3 downto 0);
begin
  with in1 select
    out1_i <= "0001" when "00",
              "0010" when "01",
              "0100" when "10",
              "1000" when "11",
              "0000" when others;
reg_proc: process(clk)
 begin
    if(rising_edge(clk)) then
           out_d <= out1_i;
    end if;
 end process reg_proc;
end dataflow;
```

Synchronous 2:4 Decoder – Test Bench

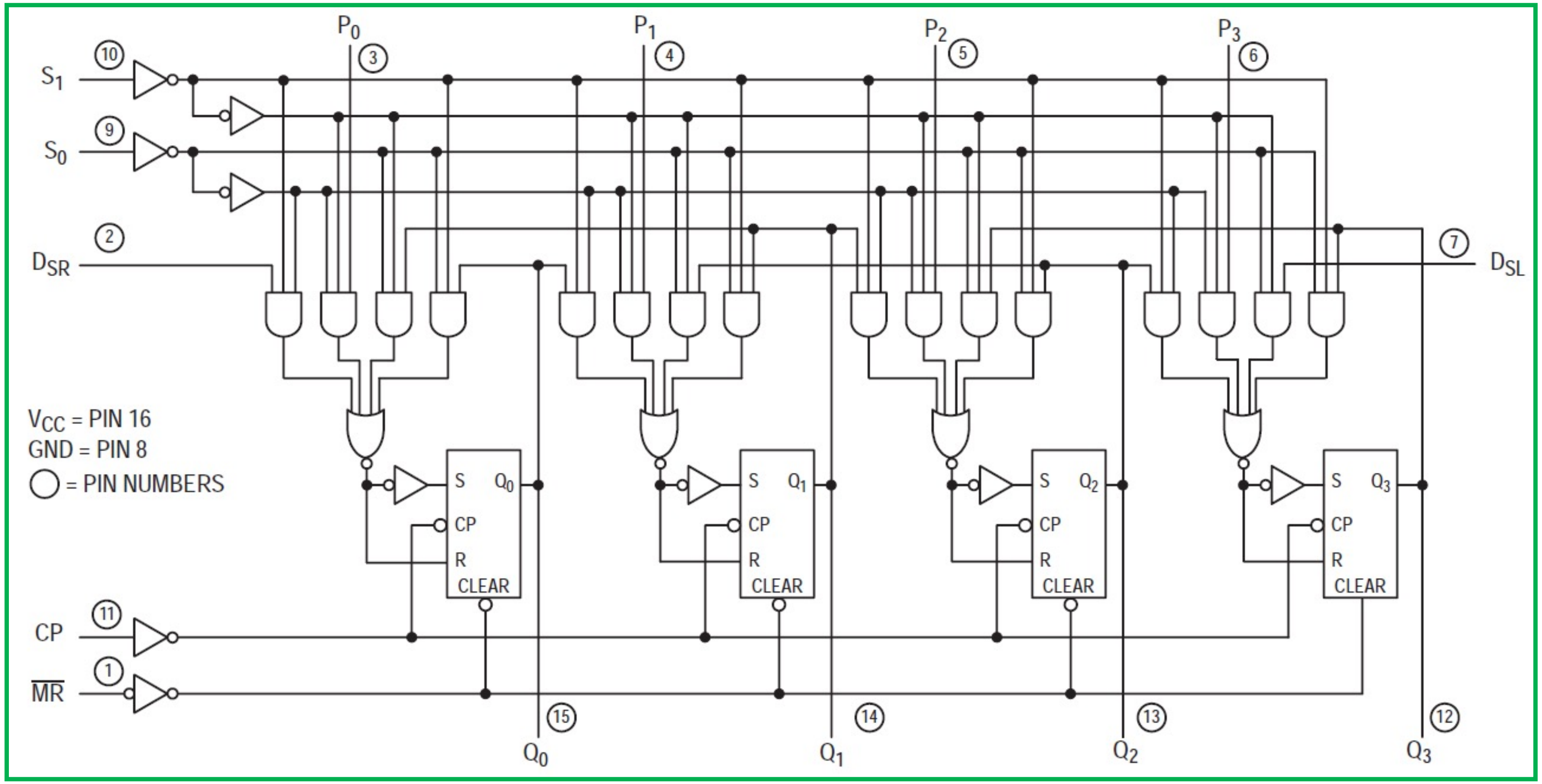
```
entity tb_synch_decode is
end tb_synch_decode ;

architecture test_bench of tb_synch_decode is
-- component declaration
…
type decoder_test is record
  in_tb : std_logic_vector(1 downto 0);
  out_tb: std_logic_vector(3 downto 0);
end record;

subtype test_array is array(natural range <>) of decoder_test;

constant test_data: test_array :=
    ("00", “0001”,
     "01", “0010",
     "10“, “0100”,
     “11”, “1000”);
```

TB – Synch. Decoder 2:4

```
begin
dut: decode port map( in_d => in_tb, out_d => out_tb);
apply_inputs: process
begin
   for j in test_data'range loop
        in_d <= test_data(j).in_tb);
        clk <= '0';
        wait for 5 ns;
        clk <= '1';
        wait for 5 ns;
         assert (out_d = test_data(j).out_tb)
               report "Output not equal to the expected value"
                    severity ERROR;
   end loop;
wait;
end process apply_inputs;
```

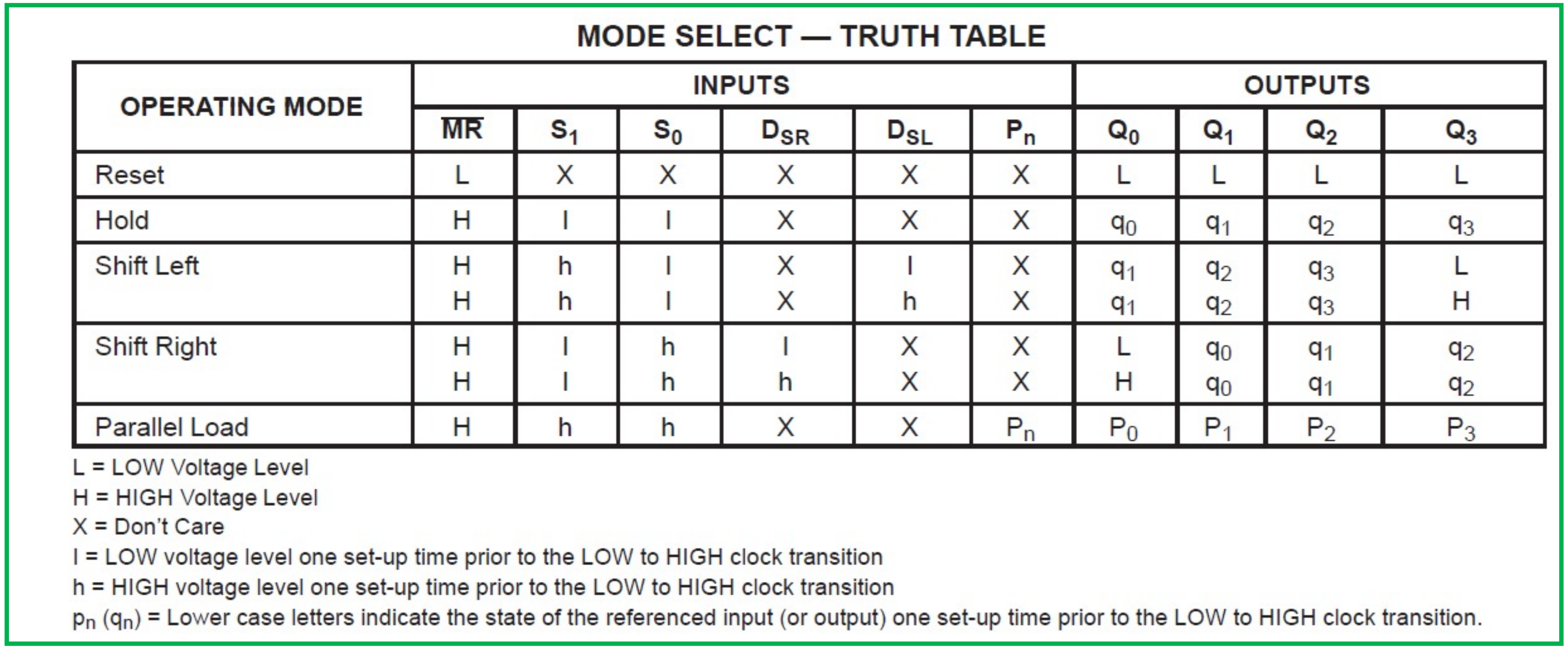
Another Example
Test Bench for a Shift Register

74x194



P0
P1
P2
P3
3
4
5
6
10
S1
9
S0
2
DSR
7
DSL
VCC = PIN 16
GND = PIN 8
= PIN NUMBERS
S
Q0
CP
R
CLEAR
S
Q1
CP
R
CLEAR
S
Q2
CP
R
CLEAR
S
Q3
CP
R
CLEAR
11
CP
1
MR
15
Q0
14
Q1
13
Q2
12
Q3

74x194



MODE SELECT — TRUTH TABLE

OPERATING MODE	INPUTS						OUTPUTS			
	$\overline{MR}$	S_1	S_0	D_{SR}	D_{SL}	P_n	Q_0	Q_1	Q_2	Q_3
Reset	L	X	X	X	X	X	L	L	L	L
Hold	H	l	l	X	X	X	q_0	q_1	q_2	q_3
Shift Left	H	h	l	X	l	X	q_1	q_2	q_3	L
	H	h	l	X	h	X	q_1	q_2	q_3	H
Shift Right	H	l	h	l	X	X	L	q_0	q_1	q_2
	H	l	h	h	X	X	H	q_0	q_1	q_2
Parallel Load	H	h	h	X	X	P_n	P_0	P_1	P_2	P_3

L = LOW Voltage Level
H = HIGH Voltage Level
X = Don't Care
l = LOW voltage level one set-up time prior to the LOW to HIGH clock transition
h = HIGH voltage level one set-up time prior to the LOW to HIGH clock transition
p_n (q_n) = Lower case letters indicate the state of the referenced input (or output) one set-up time prior to the LOW to HIGH clock transition.

74x194 – VHDL Description

```
--
-- example of a 4-bit shift register
-- LS194 4-bit bidirectional universal shift register
-- Based on the 74LS194 true table
--       MODE      -            INPUTS     -      OUTPUTS     |
--                 | MR\ S1 S0 Dsr Dsl Pn | Q0  Q1  Q2  Q3  |
-- Reset           |  L   X  X  X   X   X |  L   L   L   L  |
-- Hold            |  H   L  L  X   X   X |  Q0  Q1  Q2  Q3 |
-- Shift Left      |  H   H  L  X   H/L X |  Q1  Q2  Q3  H/L|
-- Shift Right     |  H   L  H  H/L X   X |  H/L Q0  Q1  Q2 |
-- Parallel Load |  H   H  H  X   X   Pn|  P0  P1  P2  P3 |
-- --------------------------------------------------------- |
--
```

74x194 – VHDL Description

```
library ieee;
use ieee.std_logic_1164.all;

-- entity declaration
entity ls194 is
   port(
      -- main clock
      clk    : in std_logic;
      -- async master reset
      mr_n   : in std_logic;
      -- shift control signals
      s0,s1  : in std_logic;
      -- shift input data
      dsr,dsl_ser: in std_logic;
      -- parallel load data
      p      : in  std_logic_vector(3 downto 0);
      -- output data
      q      : out std_logic_vector(3 downto 0));
end ls194;
```

74x194 – VHDL Description

```
architecture behav of ls194 is
signal temp: std_logic_vector(3 downto 0);
signal ctrl: std_logic_vector (1 downto 0);
begin
ctrl <= s0 & s1;
shift_proc: process(clk, mr_n)
begin
    if (mr_n = '0') then
        temp <= (others => '0');
     elsif (rising_edge(clk)) then
       case ctrl is
          when "11"   => temp  <= p;
         when "10"   => temp  <= dsr & temp(3 downto 1);
         when "01"   =>  temp <= temp(2 downto 0) & dsl;
          when others => temp  <= temp;
       end case;
     end if;
end process shift_proc;
q <= temp;
end behav;
```

74x194 – Test Bench

```
-- example of test bench to test the ls194
library ieee;
use ieee.std_logic_1164.all;
entity test_bench is
end test_bench;
architecture tb of test_bench is
-- internal signal declarations
signal clk_tb: std_logic:= ‘1’;
signal s0_tb, s1_tb, mr_tb_n, dsr_tb, dsl_tb: std_logic:= '0';
signal p_tb, q_tb : std_logic_vector (3 downto 0);
```

74x194 – Test Bench

```vhdl
-- constant declarations
constant clk_period: time := 200 ns;

begin
-- component instantiation
U1: work.entity.ls194 port map(
        clk  => clk_tb,
        mr_n => mr_tb_n,
        s0   => s0_tb,
        s1   => s1_tb,
        dsr  => dsr_tb,
        dsl  => dsl_tb,
        p    => p_tb,
        q    => q_tb);

-- clock generation
clk_tb <= not clk_tb after clk_period/2;
```

74x194 – Test Bench

```
main_proc: process
begin
  -- check initialization (reset)
  wait for 10 ns;
  assert q_tb = "0000"
      report " Initialization Error "
        severity ERROR;
  wait for 2 * clk_period;
  mr_tb_n <= '1';

   -- check synchronous load
   s0_tb <= '1';
   s1_tb <= '1';
   p_tb  <= "0110";
   wait for clk_period;
```

74x194 – Test Bench

```vhdl
-- wait until falling edge clk_tb
wait until clk_tb = '0';

assert q_tb = "0110"
    report " Load Error "
      severity ERROR;

-- check shift left
s0_tb <= '0';
-- wait until falling edge clk_tb
wait until clk_tb = '0';

assert q_tb = "1100"
    report " Error: Shift left Failed "
      severity ERROR;

wait for 3 * clk_period;
```

74x194 – Test Bench

```
  -- three more shift left
  for i in 0 to 2 loop
     if i = 1 then
        dsl_tb <= ‘0’;
      else
       dsl_tb <= ‘1’;
     end if;
     wait until clk_tb = ‘0’;
   end loop;

  assert q_tb = "0101"
        report " Error: serial left shift failed "
          severity ERROR;

   wait for 5 * clk_period;

   wait until clk_tb = ‘0’;
```

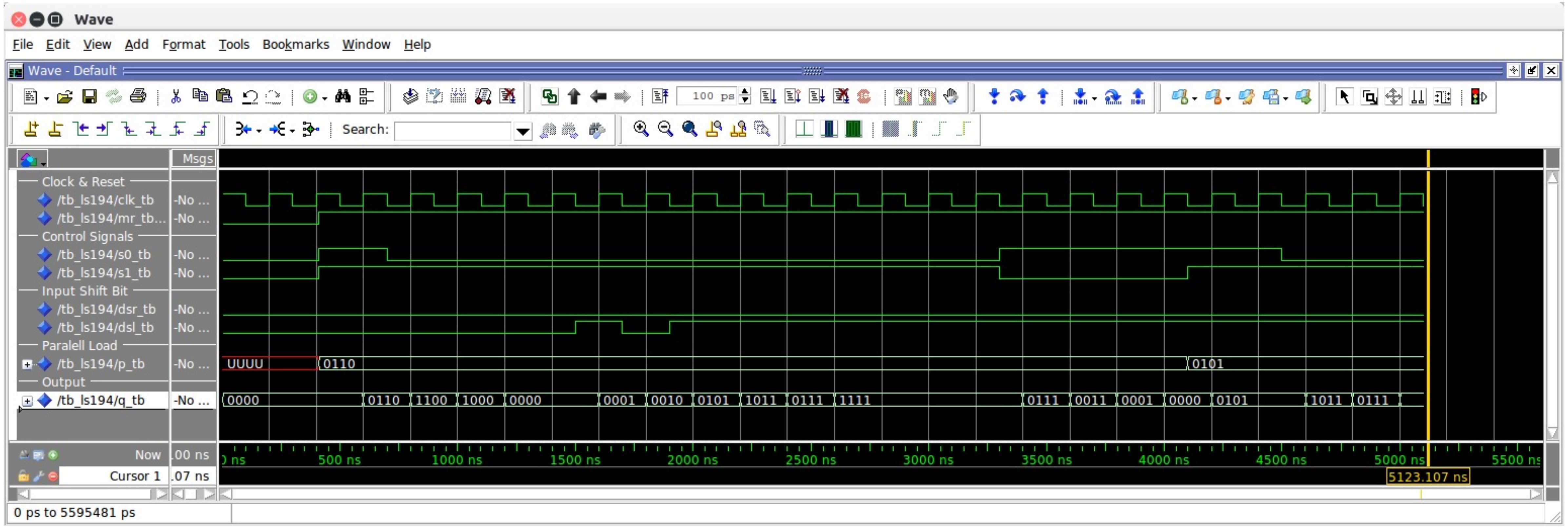
74x194 – Test Bench

```
  -- shift right
   s0_tb <= ‘1’;
   s1_tb <= ‘0’;
   wait for c3 * clk_period;
   assert q_tb = "0001"
      report " Error: serial left shift failed "
        severity ERROR;
   wait for clk_period;
   -- change load value
   s0_tb <= ‘1’;
   s1_tb <= ‘1’;
   p_tb  <= “0101”;
   wait for clk_period;
```

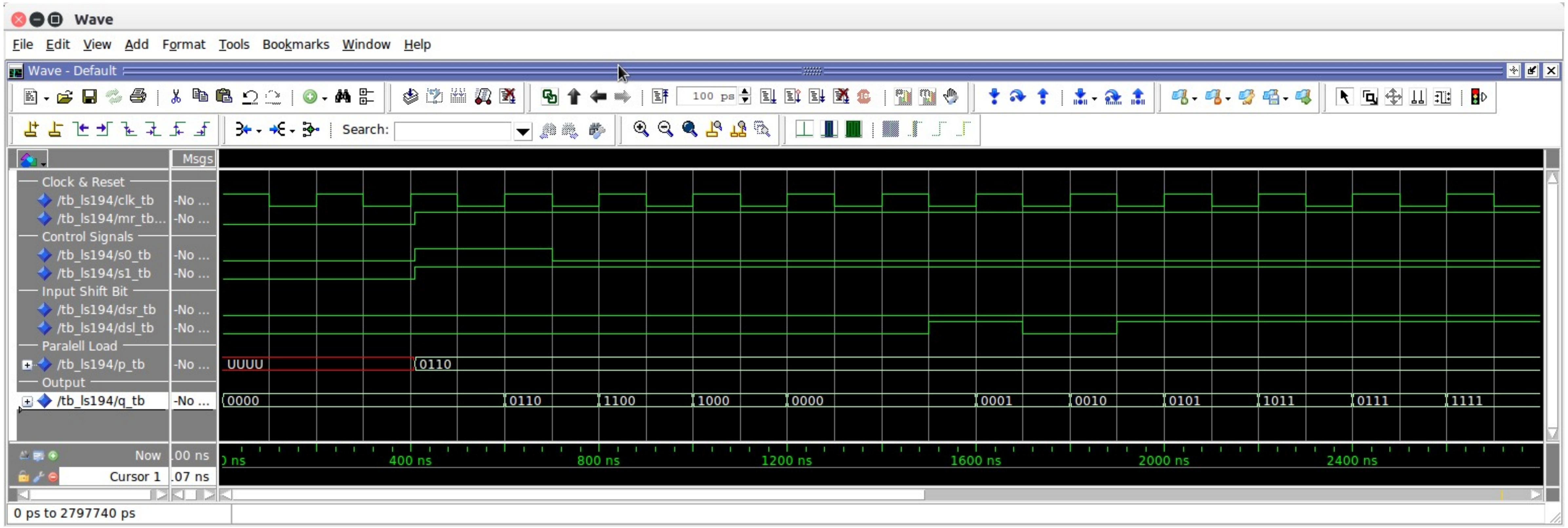
74x194 – Test Bench

```
    -- wait until falling edge clock
    wait until clk_tb = ‘0’;
    assert q_tb = “0101"
      report " Error: load failure. .  "
         severity ERROR;
    wait for clk_period;
    -- check left shift
    s0_tb <= ‘0’;
    s1_tb <= ‘1’;
    wait for 3 *clk_period;
   assert q_tb = “0111"
      report " Error: shift left failure. .  "
         severity ERROR;
end process main_proc;
end tb;
```

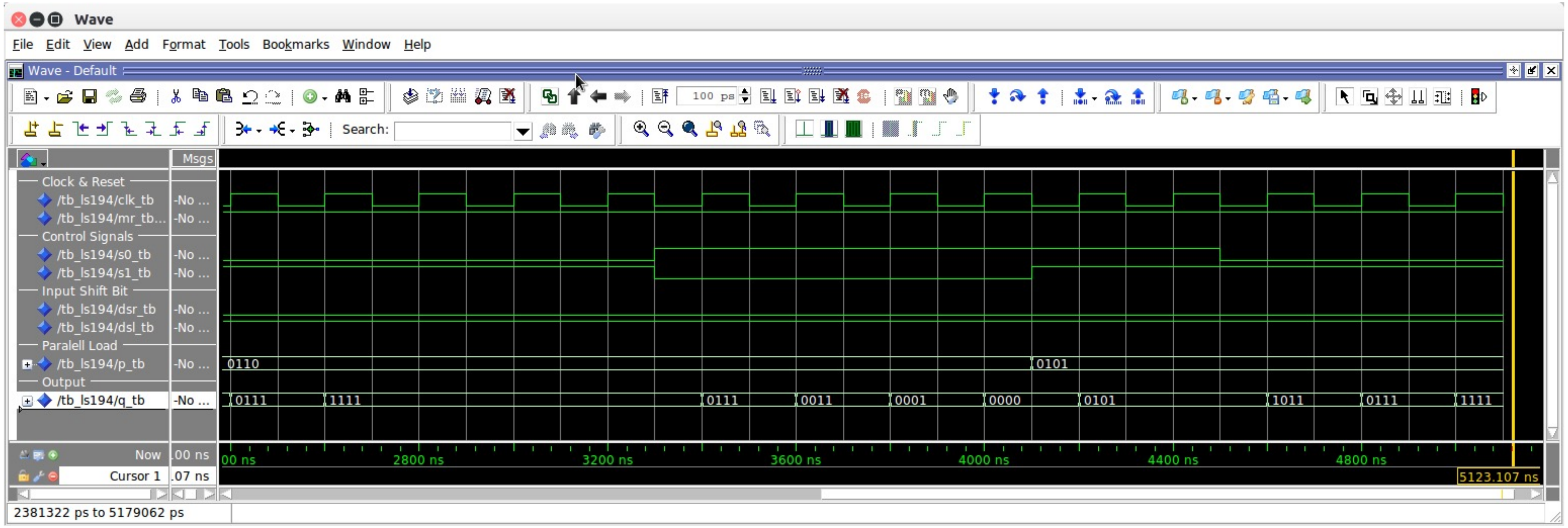
Simulation Waveforms - 1



Simulation Waveforms – 1a



Simulation Waveforms – 1b



Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Search:
Msgs
Clock & Reset
/tb_ls194/clk_tb -No ...
/tb_ls194/mr_tb... -No ...
Control Signals
/tb_ls194/s0_tb -No ...
/tb_ls194/s1_tb -No ...
Input Shift Bit
/tb_ls194/dsr_tb -No ...
/tb_ls194/dsl_tb -No ...
Paralell Load
/tb_ls194/p_tb -No ...
0110
0101
Output
/tb_ls194/q_tb -No ...
0111
1111
0111
0011
0001
0000
0101
1011
0111
1111
Now .00 ns
Cursor 1 .07 ns
00 ns
2800 ns
3200 ns
3600 ns
4000 ns
4400 ns
4800 ns
5123.107 ns
2381322 ps to 5179062 ps