Test Benches

Module 8

Overview

• We have concentrated on VHDL for synthesis
• Can also use VHDL as a test language
• Very important to conduct comprehensive verification on your design
• To simulate your design you need to produce an additional ENTITY and ARCHITECTURE design.
  – Usually referred to as a TEST BENCH
    • Not hardware, just additional VHDL!
Test Bench

- A model used to exercise and verify the correctness of a hardware model
- Has three main purposes
  - to generate stimulus for simulation
  - to apply this stimulus to the entity under test and to collect output responses
  - to compare output responses with expected values
- Test Bench should be created by a different engineer than the one who created the synthesizable VHDL
Test Bench Overview

- A Test Bench consists of
  - Entity
    - has no ports (empty entity header)
  - Architecture
    - declares, instantiates, and wires together the driver model and the model under test
    - driver model provides stimulus and verifies model responses
  - Configuration
    - specify the test case of the driver model and the version of the model under test

**Test Bench - general structure**

```vhdl
ENTITY test_bench IS
END test_bench;

ARCHITECTURE tbl OF test_bench IS
  test_component declaration
  internal signal declarations
BEGIN
  UUT: test_component instantiation
  signals to generate stimulus
  assert statements to verify responses
END tbl;
```

```vhdl
test_component

ina

inb

inc

out1
```
Test Bench - alternative structure

```vhdl
ENTITY test_bench IS
END test_bench;
```

```
Component: driver_model

ina
inb
inc
out1

Component: model_under_test
```

Test Bench - template

```vhdl
ENTITY test_bench IS
END test_bench;

ARCHITECTURE test-fixture OF test_bench IS

    COMPONENT driver_model
        END COMPONENT;

    COMPONENT model_under_test
        END COMPONENT;

    SIGNAL -- wires to connect together

BEGIN

    -- component instantiations

END test_fixture;
```
VHDL for Test Generation

- We need to look at new VHDL constructs to support our test bench design
- Modeling Time
-Generating Waveforms
- The Assert Statement
- NOTE: The following examples are for testing – NOT for synthesis

Modeling Time (not for synthesis)

- Example: 2 input AND gate

\[
c <= a \ AND \ b;
\]

- `c` changes value immediately after a change on the inputs `a` or `b`
Adding Propagation Delays

ARCHITECTURE behav OF and_gate_model IS
BEGIN
    PROCESS(a, b)
    BEGIN
        c <= a AND b;                -- instantaneous change
        -- adding "AFTER" to add delay
        c <= a AND b AFTER 10 ns; -- same tphl and tplh
        -- If we need different propagation delays:
        IF a = '1' AND b = '1' THEN
            c <= '1' AFTER 9 ns -- tplh
        ELSE
            c <= '0' AFTER 10 ns -- tphl
        END IF;
    END PROCESS;
END behav;

Adding Propagation Delays (cont’d)

- This is called the **Inertial Delay Model** (default)
- Models delays of most switching circuits
- If input pulse is shorter than the time expression (switching time of circuit) then no change occurs on the output.
Transport Delay Model (only for modeling)

- Transport delay is an optional delay model for signal assignment.
- Any pulse is transmitted, no matter how short its duration.

```
IF a = '1' AND b = '1' THEN
  c <= TRANSPORT '1' AFTER 9 ns  -- tplh
ELSE
  c <= TRANSPORT '0' AFTER 10 ns  -- tphl
END IF;
```

Generating Arbitrary Waveforms

- It is possible to assign multiple values to a signal

```
a <= '0', '1' AFTER 8 ns, '0' AFTER 13 ns, '1' AFTER 50 ns;
```

- When this signal assignment is executed (at time T) it causes four events to be scheduled for signal a
  - '0' at T
  - '1' at T + 8
  - '0' at T + 13
  - '1' at T + 50
Waveform Examples – creating clocks

ARCHITECTURE testbench OF example IS
    SIGNAL clk : std_logic := '0'; -- initialize signals
BEGIN
    clk <= NOT clk AFTER 10 ns;
    -- creates periodic waveform on signal clk
    -- with a period of 20 ns

    10 20 30 40 50

Jim Duckworth, WPI 15 Test Benches - Module 8a

Waveform Examples – more clocks

PROCESS -- Note: no sensitivity list
BEGIN
    clk <= '0';
    WAIT FOR 20 ns;
    clk <= '1';
    WAIT FOR 12 ns;
END PROCESS;

-- OR

BEGIN
    clk <= '1' AFTER 20 ns WHEN clk = '0' ELSE
        '0' AFTER 12 ns;

    20 32 52 64 84 96

Jim Duckworth, WPI 16 Test Benches - Module 8a
Waveform Examples – more clocks

- Example of two-phase clock

```vhdl
SIGNAL clk1, clk2 : std_logic := '0';

two_phase: PROCESS
BEGIN
  clk1 <= '0', '1' AFTER 5 ns, '0' AFTER 20 ns;
  clk2 <= '0', '1' AFTER 10 ns, '0' AFTER 30 ns;
  WAIT FOR 35 ns;
END PROCESS two_phase;
```

Assert Statement

- How do we check outputs?
- Assert statement is used to report information
- Assert statement checks a Boolean expression
  - if true, the statement does nothing
  - if false, the report string is output
- optional severity level to output string
  - note
  - warning
  - error
  - failure (will stop simulator)
Test Bench Example

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY test_bench IS
END test_bench;

ARCHITECTURE test_fixture OF test_bench IS
  COMPONENT and_2
    PORT (a, b : IN std_logic;
           c : OUT std_logic);
  END COMPONENT;

  -- configuration statement – may be required by simulator
  -- FOR ALL: and_2 USE ENTITY work.and2;

  SIGNAL ina, inb, out1 : std_logic; -- local signal declarations

BEGIN
  -- make copy of AND2 component to test
  UUT: and_2 PORT MAP (ina, inb, out1);

  -- create stimulus patterns
  ina <= '0', '1' AFTER 50 ns, '0' AFTER 80 ns;
inb <= '0', '1' AFTER 30 ns, '0' AFTER 120 ns;

  -- check system responses
  PROCESS
  BEGIN
  WAIT FOR 60 ns;
  ASSERT (out1 = '1')  -- if false issues report string
  REPORT "Output of AND gate incorrect at 60 ns"
  SEVERITY NOTE;
  END PROCESS;

END test_fixture;

Test Bench Example (cont’d)

BEGIN
  -- make copy of AND2 component to test
  UUT: and_2 PORT MAP (ina, inb, out1);

  -- create stimulus patterns
  ina <= '0', '1' AFTER 50 ns, '0' AFTER 80 ns;
inb <= '0', '1' AFTER 30 ns, '0' AFTER 120 ns;

  -- check system responses
  PROCESS
  BEGIN
  WAIT FOR 60 ns;
  ASSERT (out1 = '1')  -- if false issues report string
  REPORT "Output of AND gate incorrect at 60 ns"
  SEVERITY NOTE;
  END PROCESS;

END test_fixture;
Test Bench Development

- Create synthesizable VHDL as usual
  - Will end up as hardware in FPGA
- Select Project => New Source = VHDL Test Bench
  - (do not use test bench waveform)
  - A skeleton for the test bench file opens with your component already declared and instantiated
  - Enter your test bench VHDL statements

- Change from Implementation to Behavioral Simulation in the sources window
- Click on Simulate Behavioral VHDL Model
  - ISim or ModelSim starts and runs your test bench

Test Bench File – automatically created
Adding input waveform and assert statements

Simulator outputs
WAIT statements – provide timing control

- Three basic forms - causes process to suspend
  - WAIT FOR time expression
    - WAIT FOR 20 ns;
  - WAIT UNTIL Boolean expression
    - WAIT UNTIL sum > 100;
  - WAIT ON sensitivity list
    - WAIT ON a, b, c; -- waits for an event to occur on signal a, b, or c

- Variations:
  - each time event on a signal occurs the expression is evaluated
    - WAIT UNTIL sum > 100 FOR 500 ns;
    - WAIT ON clk FOR 17 ms;

Test Bench Example - shift register

- LS194 component (shift register) under test
- Start with synthesizable VHDL description

```vhdl
-- example of a 4-bit shift register
-- LS194 4-bit bidirectional universal shift register
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY LS194 IS
    PORT(clk, n_clk, s0, s1, sr_ser, sl_ser : IN std_logic;
         abcd : IN std_logic_vector(3 DOWNTO 0);
         q : OUT std_logic_vector(3 DOWNTO 0));
END LS194;
```
LS194 Component (cont’d)

ARCHITECTURE behav OF LS194 IS
  SIGNAL temp : std_logic_vector(3 DOWNTO 0);
BEGIN
  PROCESS(clk, n_clr)
  BEGIN
    IF n_clr = '0' THEN -- asynchronous clear
      temp <= "0000";
    ELSIF clk'EVENT AND clk = '1' THEN
      IF s0 = '1' AND s1 = '1' THEN -- synch load
        temp <= abcd;
      ELSIF s0 = '1' AND s1 = '0' THEN -- shift right
        temp <= sr_ser & temp(3 DOWNTO 1);
      ELSIF s0 = '0' AND s1 = '1' THEN -- shift left;
        temp <= temp(2 DOWNTO 0) & sl_ser;
      ELSE -- inhibit mode
        temp <= temp;
      END IF;
    END IF;
  END PROCESS;
  q <= temp;
END behav;

LS194 – Synthesizable VHDL
Test Bench for shift register

-- example of test_bench to test operation of '194 shift register

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY test_bench IS
END test_bench;

ARCHITECTURE test_fixture OF test_bench IS
  COMPONENT ls194
    PORT(clk, n_clr, s0, sl_ser, sr_ser : IN std_logic;
         abcd, q : std_logic_vector(3 DOWNTO 0));
  END COMPONENT;

  FOR all: ls194 USE ENTITY work.ls194; -- configuration

  -- define internal signals for connecting ls194 to driver
  SIGNAL clk, s0, sl, n_clr, sr_ser, sl_ser : std_logic := '0';
  SIGNAL abcd, q : std_logic_vector(3 DOWNTO 0);
END test_fixture;
Test Bench (cont’d)

BEGIN
-- instantiate ls194 shift register component
shift_reg:
ls194 PORT MAP(clk, n_clr, s0, s1, sr_ser, sl_ser, abcd, q);
clk <= NOT clk AFTER 50 ns; -- create system clock
PROCESS
BEGIN
WAIT FOR 10 ns;
ASSERT q = "0000"
REPORT "ERROR: clear failed"
SEVERITY error;
WAIT FOR 20 ns;
n_clr <= '1';

-- check synchronous load
s0 <= '1';
s1 <= '1';
abcd <= "0010";
WAIT UNTIL clk = '0'; -- first falling edge

ASSERT q = "0010"
REPORT "ERROR: load failed"
SEVERITY error;

-- now check shift left
s0 <= '0';
WAIT UNTIL clk = '0'; -- next falling clock edge

ASSERT q = "0011"
REPORT "ERROR: shift left failed"
SEVERITY error;
Test Bench (cont’d)

FOR i IN 0 TO 2 LOOP  -- three more shift lefts
  IF i = 1 THEN
    sl_ser <= '1';
  ELSE
    sl_ser <= '0';
  END IF;
  WAIT UNTIL clk = '0';
END LOOP;

ASSERT q = "1010"
REPORT "ERROR: serial shift left failed"
SEVERITY error;

WAIT;  -- suspend
END PROCESS;
END test_fixture;
Test Bench Simulation

- **Error: ERROR: shift left failed**
- # Time: 200 ns Iteration: 0 Instance /ls194_ls194_tb_vhd_tb
- **Error: ERROR: serial shift left failed**
- # Time: 500 ns Iteration: 0 Instance /ls194_ls194_tb_vhd_tb

New Test Bench Design

-- creating a separate driver for the test bench

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY ls194_driver IS
  PORT(clk : BUFFER std_logic := '0'; -- notice buffer
    n_clr, s0, s1, sr_ser, sl_ser : OUT std_logic := '0';
    abcd : OUT std_logic_vector(3 DOWNTO 0);
    q : IN std_logic_vector(3 DOWNTO 0));
END ls194_driver;

ARCHITECTURE behav OF ls194_driver IS
BEGIN
  clk <= NOT clk AFTER 50 ns; -- create system clock

  PROCESS
    BEGIN
      WAIT FOR 10 ns;
      ASSERT q = "0000"
New Test Bench

ENTITY test_bench2 IS
END test_bench2;

ARCHITECTURE test_fixture OF test_bench2 IS

COMPONENT ls194
PORT(clk, n_clr, s0, s1, sr_ser, sl_ser : IN std_logic;
    abcd : IN std_logic_vector(3 DOWNTO 0);
    q : OUT std_logic_vector(3 DOWNTO 0));
END COMPONENT;

COMPONENT ls194_driver
PORT(clk : BUFFER std_logic;
    n_clr, s0, s1, sr_ser, sl_ser : OUT std_logic;
    abcd : OUT std_logic_vector(3 DOWNTO 0);
    q : IN std_logic_vector(3 DOWNTO 0));
END COMPONENT;

FOR all: ls194 USE ENTITY work.ls194; -- configuration
FOR all: ls194_driver USE ENTITY work.ls194_driver;

-- define internal signals for connecting ls194 to driver
SIGNAL clk, s0, s1, n_clr, sr_ser, sl_ser : std_logic;
SIGNAL abcd, q : std_logic_vector(3 DOWNTO 0);

BEGIN
-- instantiate ls194 shift register component
shift_reg: ls194 PORT MAP(clk, n_clr, s0, s1, sr_ser, sl_ser, abcd, q);
driver: ls194_driver PORT MAP(clk, n_clr, s0, s1, sr_ser, sl_ser, abcd, q);
END test_fixture;

New Test Bench Structure

ENTITY test_bench IS
END test_bench;

Component: LS194_driver

Component: LS194

clk

S0, s1

abcd

q