

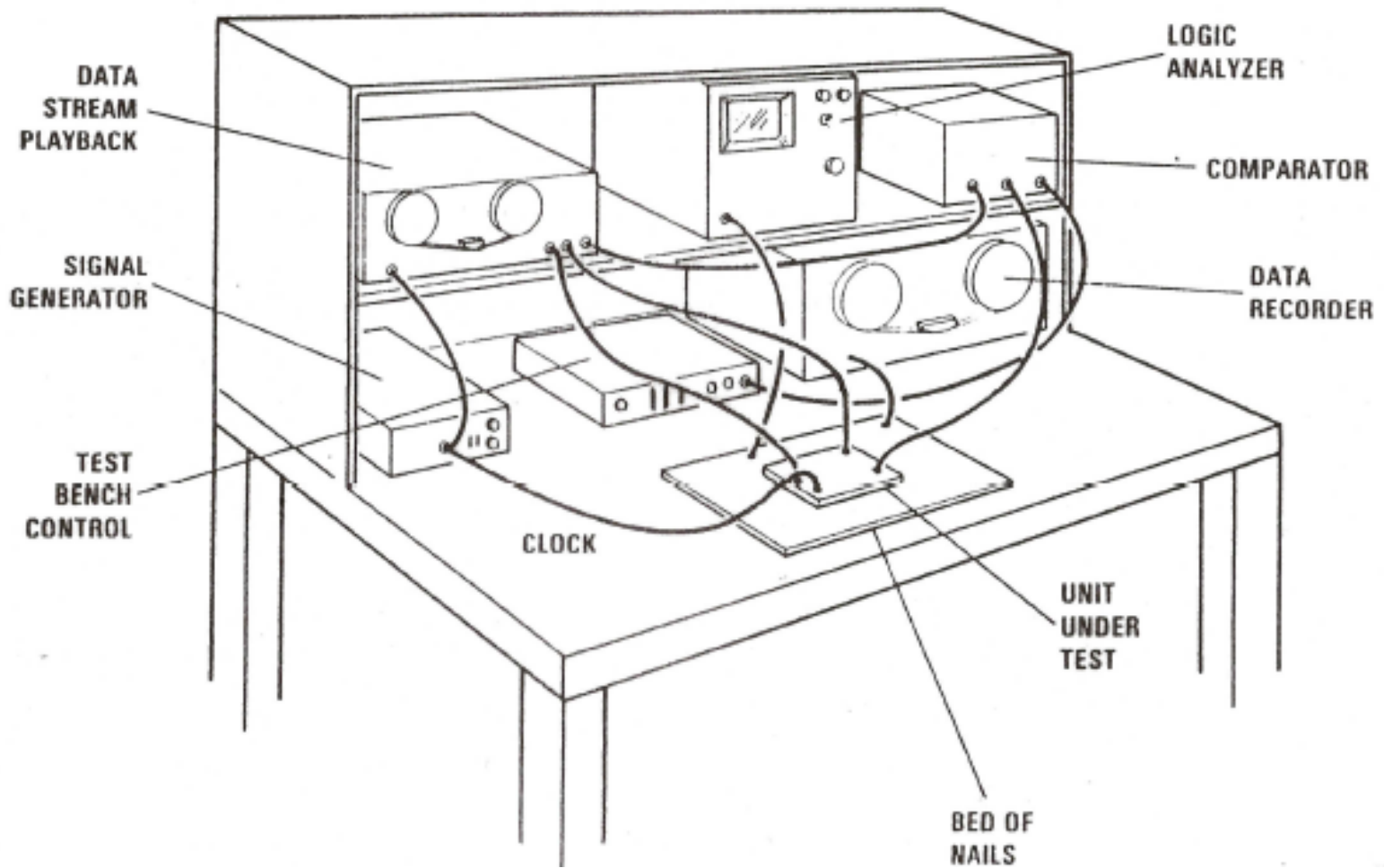


VHDL Simulation



Testbench Design

The Test Bench Concept



Project simulations

1. Behavioral/RTL – verify functionality

- ▶ Model in VHDL/Verilog
- ▶ Drive with “force file” or testbench

2. Post-Synthesis

- ▶ Synthesized gate-level VHDL/Verilog netlist
- ▶ Technology-specific VHDL/Verilog gate-level models
- ▶ Optional SDF file (from synthesis) for timing
- ▶ Drive with same force file/testbench as in (1)

3. Post-Layout

- ▶ Netlist back-annotated with extracted capacitances for accurate delays



Example: modulo-7 counter

- ▶ **VHDL Model** (*modulo7.vhd*)
 - ▶ Create working library: *vlib work*
 - ▶ Map the lib name: *vmap work work*
 - ▶ Compile: *vcom modulo7.vhd*
 - ▶ Simulate: *vsim modulo7(behavior)*
- ▶ **Simulation-control**
 - ▶ Modelsim “macro” file (*mod7.do*)
 - ▶ Testbench (*VHDL or Verilog*)
- ▶ **ModelSim results**
 - ▶ List(table) and/or Waveform (logic analyzer)



-- modulo7.vhd parallel-load modulo-7 synchronous counter

```
library ieee; use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity modulo7 is
port(reset,count,load,clk: in std_logic;
      l: in std_logic_vector(2 downto 0);
      Q: out std_logic_vector(2 downto 0));
end modulo7;
architecture Behave of modulo7 is
  signal Q_s: unsigned(2 downto 0);
begin
  process (reset,clk) begin
    if (reset='0') then
      Q_s <= "000";
    elsif (clk'event and (clk='1')) then
      if (count = '1') and (Q_s = "110") then
        Q_s <= "000";
      elsif (count='1') then
        Q_s <= Q_s + 1;
      elsif (load='1') then
        Q_s <= UNSIGNED(l);
      end if;
    end if;
  end process;
  Q<=STD_LOGIC_VECTOR(Q_s);
end Behave;
```



Test stimulus:

Modelsim “do” file: mod7.do

```
add wave /clk /reset /count /load
add wave -hex /I /Q
add list /clk /reset /count /load
add list -hex /I /Q
force /clk 0 0 ns, 1 20 ns -repeat 40 ns
force /I 101 0 ns, 011 400 ns
force /reset 1 0 ns, 0 10 ns, 1 20 ns
force /count 0 0 ns, 1 90 ns, 0 490 ns
force /load 0 0 ns, 1 30 ns, 0 70 ns
run 600 ns
```



Results in list format

ModelSim SE 6.1f

File Edit View Format Compile Simulate Add Tools Window Help

100 ps

Objects

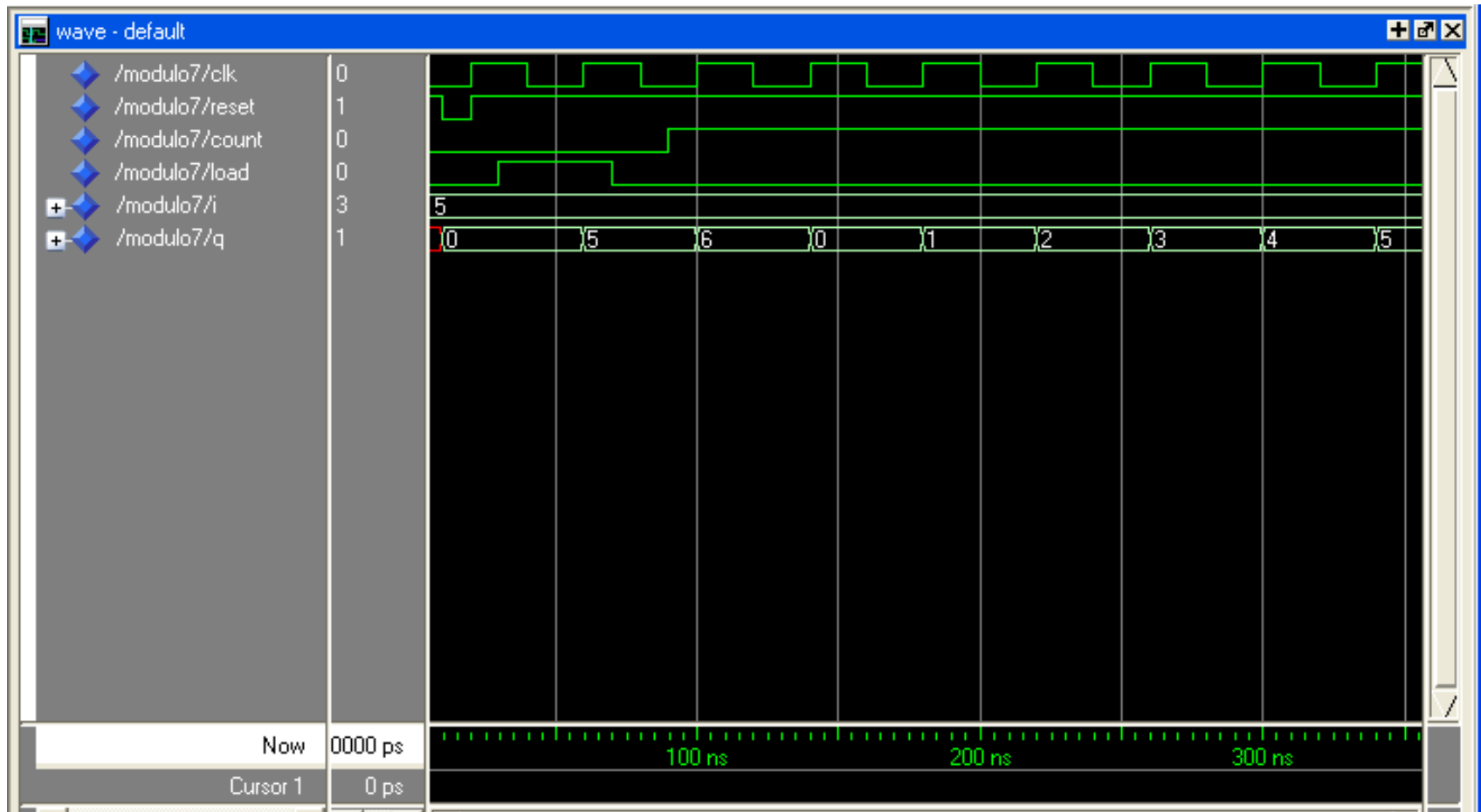
Name	Value
clk	0
count	0
i	011
load	0
q	001
q_s	001
reset	1

Note "delta" delays for behavioral model.

list

ps	delta	/modulo7/clk	/modulo7/q	/modulo7/reset	/modulo7/count	/modulo7/load	/modulo7/i
0	+0	U	U	U	U	X	X
0	+1	0	1	0	0	5	X
10000	+0	0	0	0	0	5	X
10000	+2	0	0	0	0	5	0
20000	+0	1	1	0	0	5	0
30000	+0	1	1	0	1	5	0
40000	+0	0	1	0	1	5	0
60000	+0	1	1	0	1	5	0
60000	+2	1	1	0	1	5	5
70000	+0	1	1	0	0	5	5
80000	+0	0	1	0	0	5	5
90000	+0	0	1	1	0	5	5
100000	+0	1	1	1	0	5	5
100000	+2	1	1	1	0	5	6
120000	+0	0	1	1	0	5	6
140000	+0	1	1	1	0	5	6
140000	+2	1	1	1	0	5	0
160000	+0	0	1	1	0	5	0

Results in wave format



Elements of a VHDL/Verilog testbench

- ▶ **Unit Under Test (UUT) – or Device Under Test (DUT)**
 - ▶ instantiate one or more UUT's
- ▶ **Stimulus of UUT inputs**
 - ▶ algorithmic
 - ▶ from arrays
 - ▶ from files
- ▶ **Checking of UUT outputs**
 - ▶ assertions
 - ▶ write to files



Instantiating the UUT

-- 32 bit adder testbench

entity adder_bench is -- no top-level I/O ports

end adder_bench;

architecture test of adder_bench is

component adder is -- declare the UUTs

port (

 X,Y: in std_logic_vector(31 downto 0);

 Z: out std_logic_vector(31 downto 0)

);

signal A,B,Sum: std_logic_vector(31 downto 0); --internal signals

begin

 UUT: adder port map (A,B,Sum); --instantiate the adder



Example – stimulating clock inputs

-- Simple 50% duty cycle clock

clk <= not clk after T ns; -- T is constant or defined earlier

-- Clock process, using “wait” to suspend for T1/T2

process begin

clk <= '1'; wait for T1 ns; -- clk high for T1 ns

clk <= '0'; wait for T2 ns; -- clk low for T2 ns

end process;

-- Clock “procedure” (define in declaration area or in a package)

procedure Clock (signal C: out bit; HT, LT:TIME) is begin

loop -- schedule “waveform” on C and suspend for period

C <= '1' after LT, '0' after LT + HT; wait for LT + HT;

end loop;

end procedure;

-- “execute” clock procedure by instantiating it in the architecture

CI: Clock (CLK, 10ns, 8 ns);



Algorithmic generation of stimulus

-- Generate test values for an 8-bit adder inputs A & B

process begin

for m in 0 to 255 loop -- all 8 bit addend values

for n in 0 to 255 loop -- all 8 bit augend values

A <= to_std_logic(m); -- apply m to adder input A

B <= to_std_logic(n); -- apply n to adder input B

wait for T ns; -- allow time for addition

assert (to_integer(Sum) = (m + n)) -- expect Sum = A + B

report "Incorrect sum" severity note;

end loop; end loop;

end process;



Sync patterns with clock transitions

-- Test 4x4 bit multiplier algorithm

process begin

for m in 0 to 15 loop;

for n in 0 to 15 loop;

A <= to_std_logic(m); -- apply multiplier

B <= to_std_logic(n); -- apply multiplicand

wait until CLK'EVENT and CLK = '1'; -- clock in A & B

wait for 1 ns; -- move next change past clock edge

Start <= '1','0' after 20 ns; -- pulse Start signal

wait until Done = '1'; -- wait for end of multiply

wait until CLK'EVENT and CLK = '1'; -- finish last clock

end loop; end loop; end process;



Reading test patterns from files

```
use std.textio.all;           -- Contains file/text support
architecture m1 of bench is begin
    signal Vec: std_logic_vector(7 downto 0); -- test vector
process
    file P: text open read_mode is "testvecs"; -- test vector file
    variable LN: line;           -- temp variable for file read
    variable LB: bit_vector(31 downto 0); -- for read function
begin
    while not endfile(P) loop -- Read vectors from data file
        readline(P, LN);      -- Read one line of the file (type "line")
        read(LN, LB);         -- Get bit_vector from line
        Vec <= to_stdlogicvector(LB); -- Vec is std_logic_vector
    end loop; end process;
```



Test vectors from an array

```
type vectors is array (1 to N) of std_logic_vector(7 downto 0);
```

```
  signal V: vectors := -- initialize vector array
```

```
  (
    "00001100", -- pattern 1
    "00001001", -- pattern 2
    "00110100", -- pattern 3
    .....,
    "00111100"  -- pattern N
  );
```

```
begin
```

```
  A <= V(i);      -- set A to ith vector
```



Check results with assertions

-- Assert statement checks for expected condition

assert (A = (B + C)) -- expect A = B+C (any boolean condition)

report "Error message"

severity NOTE;

-- Print "Error message" if assert condition FALSE
(condition is not what we expected)

-- Specify one of four severity levels:

NOTE, WARNING, ERROR, FAILURE

-- Modelsim allows selection of severity level that should
halt the simulation

-- Severity level NOTE generally should not stop simulation



Check timing constraints

-- Tsu for flip flop D input before clock edge is 2ns

```
assert not (CK'stable and (CK = '1') and not D'stable(2ns))  
  report "Setup violation: D not stable for 2ns before CK";
```

-- DeMorgan equivalent

```
assert CK'stable or (CK = '0') or D'stable(2ns)  
  report "Setup violation: D not stable for 2ns before CK";
```



Testbench: *modulo7_bench.vhd*

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY modulo7_bench is end modulo7_bench;

ARCHITECTURE test of modulo7_bench is
    component modulo7
        PORT (reset,count,load,clk: in std_logic;
             I: in std_logic_vector(2 downto 0);
             Q: out std_logic_vector(2 downto 0));
    end component;
    for all: modulo7 use entity work.modulo7(Behave);
    signal clk : STD_LOGIC := '0';
    signal res, cnt, ld: STD_LOGIC;
    signal din, qout: std_logic_vector(2 downto 0);

begin
    -- instantiate the component to be tested
    UUT: modulo7 port map(res,cnt,ld,clk,din,qout);
```

Alternative
to “do” file

Continue on
next slide



Testbench: modulo7_bench.vhd

qint = expected outputs of UUT

```
clk <= not clk after 10 ns;
```

```
PI: process
```

```
  variable qint: UNSIGNED(2 downto 0);
```

```
  variable i: integer;
```

```
begin
```

```
  qint := "000";
```

```
  din <= "101"; res <= '1';
```

```
  cnt <= '0'; ld <= '0';
```

```
  wait for 10 ns;
```

```
  res <= '0';    --activate reset for 10ns
```

```
  wait for 10 ns;
```

```
  assert UNSIGNED(qout) = qint
```

```
    report "ERROR Q not 000"
```

```
    severity WARNING;
```

```
  res <= '1';    --deactivate reset
```

```
  wait for 5 ns; --hold after reset
```

```
  ld <= '1';    --enable load
```

```
  wait until clk'event and clk = '1';
```

```
  qint := UNSIGNED(din); --loaded value
```

```
  wait for 5 ns;    --hold after load
```

```
  ld <= '0';        --disable load
```

```
  cnt <= '1';      --enable count
```

```
  for i in 0 to 20 loop
```

```
    wait until clk'event and clk = '1';
```

```
    assert UNSIGNED(qout) = qint
```

```
      report "ERROR Q not Q+1"
```

```
      severity WARNING;
```

```
    if (qint = "110") then
```

```
      qint := "000";    --roll over
```

```
    else
```

```
      qint := qint + "001"; --increment
```

```
    end if;
```

```
  end loop;
```

```
end process;
```

Print message if incorrect result