Steps of the Design Process

1. Text description
2. Interface
3. Pseudocode
4. Block diagram of the Datapath
5. Interface with the division into the Datapath and the Controller
6. ASM chart of the Controller
7. RTL VHDL code of the Datapath, the Controller, and the Top Unit
8. Testbench of the Datapath, the Controller, and the Top Unit
9. Functional simulation and debugging
10. Synthesis and post-synthesis simulation
11. Implementation and timing simulation
12. Experimental testing

Required reading

- Sundar Rajan, Essential VHDL: RTL Synthesis Done Right
  Chapter 14, starting from “Design Verification”

Simple Testbench

Processes Generating Input Stimuli → Design Under Test (DUT)

Outputs Observed as Timing Waveforms

Advanced Testbench

Processes Generating Input Stimuli → Design Under Test (DUT) → Process Comparing Actual Outputs vs. Expected Outputs

Testvector(s) → Yes/No Design Correct/Incorrect

Possible Sources of Expected Outputs

Source of Representative Inputs → VHDL Design
Manual Calculations or Reference Software Implementation (C, Java, Matlab)

Actual Outputs = ?

Expected Outputs
**Test vectors**

Set of pairs:  \{Input i, Expected Output i\}

- Input 1, Expected Output 1
- Input 2, Expected Output 2
- Input N, Expected Output N

Test vectors can be:
- defined in the testbench source file
- stored in a data file

---

**Assert**

Assert is a **non-synthesizable** statement whose purpose is to write out messages on the screen when problems are found during simulation.

Depending on the **severity of the problem**, The simulator is instructed to continue simulation or halt.

---

**Assert - syntax**

```
ASSERT condition
[REPORT "message"]
[SEVERITY severity_level ];
```

The message is written when the condition is FALSE.

Severity_level can be:
- Note, Warning, Error (default), or Failure.

---

**Assert – Examples (1)**

```
assert initial_value <= max_value
   report "initial value too large"
   severity error;

assert packet_length /= 0
   report "empty network packet received"
   severity warning;

assert false
   report "initialization complete"
   severity note;
```

---

**Assert – Examples (2)**

```
-- suppose VHDL model is supposed to generate data
-- with even parity with ack_test:

wait until ack_test = '1';
parity := test_data(3) XOR test_data(2) XOR test_data(1) XOR test_data(0);
assert parity = '0'
   report "Parity Error"
   severity error;
R_test <= '0';
```
Assert – Examples (3)
-- used in modeling flip-flops to check for timing problems
check: process
begin
  wait until clk'event and clk='1';
  assert D'stable(setup_time)
    report "Setup Time Violation"
    severity error;
  wait for hold_time;
  assert D'stable(hold_time)
    report "Hold Time Violation"
    severity error;
end process check;

Report - syntax
REPORT "message"
[SEVERITY severity_level ];
The message is always written.
Severity_level can be:
Note (default), Warning, Error, or Failure.

Report - Examples
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity example_1_tb is
end example_1_tb;
arquitectura behavioral of example_1_tb is
  signal clk : std_logic := '0';
begin
  clk <= not clk after 100 ns;
  process
    begin
      wait for 1000 ns;
      report "Initialization complete";
      report  "Current time = " & time'image(now);
      wait for 1000 ns;
      report "SIMULATION COMPLETED"  severity failure;
    end process;
end behavioral;

Generating reports in the message window
reports: process(clk_trigger) begin
  if (clk_trigger = '0' and clk_trigger'EVENT) then
    case segments is
      when seg_0 => report time'image(now) & "0 is displayed" ;
      when seg_1 => report time'image(now) & "1 is displayed" ;
      when seg_2 => report time'image(now) & "2 is displayed" ;
      when seg_3 => report time'image(now) & "3 is displayed" ;
      when seg_4 => report time'image(now) & "4 is displayed" ;
      when seg_5 => report time'image(now) & "5 is displayed" ;
      when seg_6 => report time'image(now) & "6 is displayed" ;
      when seg_7 => report time'image(now) & "7 is displayed" ;
      when seg_8 => report time'image(now) & "8 is displayed" ;
      when seg_9 => report time'image(now) & "9 is displayed" ;
      when others => report "Invalid segment" ;
    end case;
  end if;
end process;
### Records

```
Records

type opcodes is (add, sub, and, or);
type reg_number is range 0 to 8;

type instruction is record
    opcode          : opcodes;
    source_reg1  : reg_number;
    source_reg2  : reg_number;
    dest_reg        : reg_number;
end record instruction;

constant add_instr_1_3 : instruction:=
  (opcode => add,
   source_reg1 | dest_reg => 1,
   source_reg2 => 3);
```

### Variables

#### Variable – Example (1)

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

ENTITY Numbits IS
  PORT ( X      : IN      STD_LOGIC_VECTOR(15 DOWNTO 0) ;
         Count   : OUT  INTEGER RANGE 0 TO 16) ;
END Numbits ;

ARCHITECTURE Behavior OF Numbits IS
BEGIN
  PROCESS(X) – count the number of bits in X equal to 1
    VARIABLE Tmp: INTEGER;
    BEGIN
      Tmp := 0;
      FOR i IN 15 DOWNTO 0 LOOP
        IF X(i) = '1' THEN
          Tmp := Tmp + 1;
        END IF;
      END LOOP;
      Count <= Tmp;
    END PROCESS;
END Behavior ;
```

#### Variable – Example (2)

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

ENTITY Numbits IS
  PORT ( X      : IN      STD_LOGIC_VECTOR(15 DOWNTO 0) ;
         Count   : OUT  INTEGER RANGE 0 TO 16) ;
END Numbits ;

ARCHITECTURE Behavior OF Numbits IS
BEGIN
  PROCESS(X) – count the number of bits in X equal to 1
    VARIABLE Tmp: INTEGER;
    BEGIN
      Tmp := 0;
      FOR i IN 15 DOWNTO 0 LOOP
        IF X(i) = '1' THEN
          Tmp := Tmp + 1;
        END IF;
      END LOOP;
      Count <= Tmp;
    END PROCESS;
END Behavior ;
```

### Variables - features

- Can only be declared within processes and subprograms (functions & procedures)
- Initial value can be explicitly specified in the declaration
- When assigned take an assigned value immediately
- Variable assignments represent the desired behavior, not the structure of the circuit
- Should be avoided, or at least used with caution in a synthesizable code

### Using Arrays of Test Vectors In Testbenches
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY sevenSegmentTB IS END sevenSegmentTB;
ARCHITECTURE testbench OF sevenSegmentTB IS
COMPONENT sevenSegment
PORT ( bcdInputs : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
seven_seg_outputs : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
); end COMPONENT;
CONSTANT PropDelay: time := 40 ns;
CONSTANT SimLoopDelay: time := 10 ns;
TYPE vector IS RECORD
bcdStimulus: STD_LOGIC_VECTOR(3 downto 0);
sevSegOut: STD_LOGIC_VECTOR(6 downto 0);
END RECORD;
CONSTANT NumVectors: INTEGER := 10;
TYPE vectorArray is ARRAY (0 TO NumVectors - 1) OF vector;
CONSTANT vectorTable: vectorArray := (
(bcdStimulus => "0000", sevSegOut => "0000001"),
(bcdStimulus => "0001", sevSegOut => "1001111"),
(bcdStimulus => "0010", sevSegOut => "0010010"),
(bcdStimulus => "0011", sevSegOut => "0000110"),
(bcdStimulus => "0100", sevSegOut => "1001100"),
(bcdStimulus => "0101", sevSegOut => "0100100"),
(bcdStimulus => "0110", sevSegOut => "0100000"),
(bcdStimulus => "0111", sevSegOut => "0001111"),
(bcdStimulus => "1000", sevSegOut => "0000000"),
(bcdStimulus => "1001", sevSegOut => "0000100")
);
SIGNAL StimInputs: STD_LOGIC_VECTOR(3 downto 0);
SIGNAL CaptureOutputs: STD_LOGIC_VECTOR(6 downto 0);
BEGIN
u1: sevenSegment PORT MAP ( bcdInputs => StimInputs,
seven_seg_outputs => CaptureOutputs);
END PROCESS;
END testbench;
FILE I/O
File I/O Example

- Example of file input/output using a counter
- Text file is vectorfile.txt
  - Has both input data and EXPECTED output data
  - Will compare VHDL output data with EXPECTED output data!

Design Under Test (1)

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY loadCnt IS
  PORT (data: STD_LOGIC_VECTOR (7 DOWNTO 0);
        load: STD_LOGIC;
        clk: STD_LOGIC;
        rst: STD_LOGIC;
        q: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
      );
END loadCnt;

Design Under Test (2)

ARCHITECTURE rtl OF loadCnt IS
SIGNAL cnt: STD_LOGIC_VECTOR (7 DOWNTO 0);
BEGIN
  counter: PROCESS (clk, rst)
  BEGIN
    IF (rst = '1') THEN
      cnt <= (OTHERS => '0');
    ELSIF (clk'event AND clk = '1') THEN
      IF (load = '1') THEN
        cnt <= data;
      ELSE
        cnt <= cnt + 1;
      END IF;
    END IF;
    q <= cnt;
  END PROCESS;
END rtl;

Test vector file (1)

#Format is Rst, Load, Data, Q
#load the counter to all 1s
0 1 11111111 11111111
#reset the counter
1 0 10101010 00000000
#perform load/increment for each bit
0 1 11111110 11111110
0 0 11111110 11111110
0 1 11111110 11111110
0 1 11111111 11111110
0 0 11111111 11111110
0 1 11111111 11111110
0 0 11111111 11111100
#check roll-over case
0 1 11111111 11111111
0 0 11111111 00000000
# End vectors

Test vector file (2)

# 0 1 11101111 11101111
# 0 1 11011111 11011111
# 0 1 10111111 10111111
# 0 0 10111111 10111111
# 0 1 01111111 01111111
# 0 0 01111111 01111111
# 0 1 11111111 11111111
# 0 1 11111111 11111111
# 0 1 11111111 11111111
# 0 0 11111111 11111111
# End vectors

Methodology to test vectors from file

Verify output is as expected:
- compare Qout (the output of the VHDL counter)
  with Qexpected (the expected value of Q from the test file)

Apply input data to counter (i.e. rst <= vRst,
load <= vLoad,
reset <= vReset,
data <= vData)

Read vector from text file into variables
(vRst, vLoad, vData, vQ)
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_textio.all;

LIBRARY std;
USE std.textio.all;

ENTITY loadCntTB IS
END loadCntTB;

ARCHITECTURE testbench OF loadCntTB IS

COMPONENT loadCnt
PORT (  
data: IN   STD_LOGIC_VECTOR (7 DOWNTO 0);
load: IN   STD_LOGIC;
clk: IN   STD_LOGIC;
rst: IN   STD_LOGIC;
q: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)  
);  
END COMPONENT;

FILE vectorFile: TEXT OPEN READ_MODE is "vectorfile.txt";

SIGNAL Data: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL Load: STD_LOGIC;
SIGNAL Rst: STD_LOGIC;
SIGNAL Qout: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL Qexpected: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL TestClk: STD_LOGIC := '0';
CONSTANT ClkPeriod: TIME := 100 ns;

BEGIN
-- Free running test clock
TestClk <= NOT TestClk AFTER ClkPeriod/2;
-- Instance of design being tested
u1: loadCnt PORT MAP (Data => Data,
load => Load,
clk => TestClk,
rst => Rst,
q => Qout);

-- File reading and stimulus application
readVec: PROCESS
VARIABLE VectorLine: LINE;
VARIABLE VectorValid: BOOLEAN;
VARIABLE vRst: STD_LOGIC;
VARIABLE vLoad: STD_LOGIC;
VARIABLE vData: STD_LOGIC_VECTOR(7 DOWNTO 0);
VARIABLE vQ: STD_LOGIC_VECTOR(7 DOWNTO 0);
VARIABLE space: CHARACTER;
BEGIN
WHILE NOT ENDFILE (vectorFile) LOOP
readline(vectorFile, VectorLine);
-- put file data into line
read(VectorLine, vRst, good => VectorValid);
NEXT WHEN NOT VectorValid;
read(VectorLine, space);
read(VectorLine, vLoad);
read(VectorLine, space);
read(VectorLine, vData);
read(VectorLine, space);
WAIT FOR ClkPeriod/4;
Rst <= vRst;
Load <= vLoad;
Data <= vData;
Qexpected <= vQ;
WAIT FOR (ClkPeriod/4) * 3;
END LOOP;

ASSERT FALSE
REPORT "Simulation complete"
SEVERITY NOTE;
WAIT;
END PROCESS;

-- Process to verify outputs
verify: PROCESS (TestClk)
variable ErrorMsg: LINE;
BEGIN
IF (TestClk'event AND TestClk = '0') THEN
IF Qout /= Qexpected THEN
write(ErrorMsg, STRING'("Vector failed "));
write(ErrorMsg, now);
writeline(output, ErrorMsg);
END IF;
END IF;
END PROCESS;
END testbench;
**Hex format**

In order to read/write data in the hexadecimal notation, replace

read with hread, and

write with hwrite

---

**Note on test file**

- This example showed a test file that had both the control commands (i.e. load, reset), and the actual data itself
- Often the test file just has the input and output vectors (and no load, reset, etc.)