The VHDL Hardware Description Language

CSEE W4840

Prof. Stephen A. Edwards

Columbia University

Why HDLs?

1970s: SPICE transistor-level netlists

An XOR built from four NAND gates

.Model P PMOS
.Model N NMOS
.Subcckt NAND A B Y VDD VSS
M1 Y A VDD VDD P
M2 Y B VDD VDD P
M3 Y A X VSS N
M4 X B VSS VSS N
.Ends

X1 A B 11 VSS 0 NAND
X2 A I1 I2 VDD 0 NAND
X3 B I1 I3 VSS 0 NAND
X4 I2 I3 Y VDD 0 NAND

Why HDLs?

1980s: Graphical schematic capture programs

Why HDLs?

1990s: HDLs and Logic Synthesis

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity ALU is
port(A, B: in std_logic_vector(1 downto 0);
    Sel: in std_logic_vector(1 downto 0);
    Res: out std_logic_vector(1 downto 0));
end ALU;

architecture behv of ALU is
begin
    process(A, B, Sel) begin
        case Sel is
            when "00" => Res <= A + B;
            when "01" => Res <= A + (not B) + 1;
            when "10" => Res <= A and B;
            when "11" => Res <= A or B;
            when others => Res <= "XX";
        end case;
    end process;
end behv;

Two Separate but Equal Languages

Verilog and VHDL

Verilog: More succinct, less flexible, really messy

Part of languages people actually use identical.
Every synthesis system supports both.

VHDL: Hierarchical Models

Ports

Component

Process

Signal

Dataflow Expression

X <= (Y = '1') and (X = "110")

Basic VHDL: Full Adder

library ieee; -- part of IEEE library
use ieee.std_logic_1164.all; -- includes std_ulogic

entity full_adder is
    port(a, b, c : in std_ulogic;
        sum, carry : out std_ulogic);
end full_adder;

architecture imp of full_adder is
begin
    sum <= (a xor b) xor c; -- combinational logic
    carry <= (a and b) or (a and c) or (b and c);
end imp;

library ieee;
use ieee.std_logic_1164.all;

entity add2 is
    port (A, B : in std_logic_vector(1 downto 0);
        C : out std_logic_vector(2 downto 0));
end add2;

architecture imp of add2 is
    component full_adder
        port (a, b, c : in std_ulogic;
            sum, carry : out std_ulogic);
    end component;

    signal carry : std_ulogic;

    begin
        bit0 : full_adder port map (
            a => A(0),
            b => B(0),
            c => '0',
            sum => C(0),
            carry => carry);
        bit1 : full_adder port map (
            a => A(1),
            b => B(1),
            c => carry,
            sum => C(1),
            carry => carry);
    end imp;

VHDL: Two-bit Counter

library ieee;
use ieee.std_logic_1164.all;

entity counter is
    port (Clk : in std_logic;
        Count : out std_logic_vector(2 downto 0));
end counter;

architecture imp of counter is
begin
    counter <= 0;
    process(Clk) begin
        if Clk'Event and Clk='1' then
            counter <= counter + 1;
        end if;
    end process;
end imp;

Four-to-one multiplexer: when...else

library ieee;
use ieee.std_logic_1164.all;

entity multiplexer_4_1 is
    port(in0, in1 : in std_logic_vector(15 downto 0);
        in2, in3 : in std_logic_vector(15 downto 0);
        s0, s1 : in std_logic;
        z : out std_logic_vector(15 downto 0));
end multiplexer_4_1;

architecture imp of multiplexer_4_1 is
begin
    z <= in0 when (s0 = '0' and s1 = '0') else
        in1 when (s0 = '1' and s1 = '0') else
        in2 when (s0 = '0' and s1 = '1') else
        in3 when (s0 = '1' and s1 = '1') else
            "XXXXXXXXXXXXXXXX";
end imp;
The VHDL Hardware Description Language – p. 19/

**A small RAM**

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

entity ram_32_4 is
  port(
    Clk : in std_logic;
    WE : in std_logic; -- Write enable
    EN : in std_logic; -- Read enable
    addr : in std_logic_vector(4 downto 0);
    di : in std_logic_vector(3 downto 0); -- Data in
    do : out std_logic_vector(3 downto 0)); -- Data out
end ram_32_4;

architecture imp of ram_32_4 is
begin
  process (Clk)
  begin
    if (Clk'event and Clk = '1') then
      if (WE = '1') then
        RAM(conv_integer(addr)) <= di;
      else
        do <= RAM(conv_integer(addr));
      end if;
    end if;
  end process;
end imp;
```

The VHDL Hardware Description Language – p. 21/

**A small ROM**

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

entity rom_32_4 is
  port(
    Clk : in std_logic;
    en : in std_logic; -- Read enable
    addr : in std_logic_vector(4 downto 0);
    data : out std_logic_vector(3 downto 0));
end rom_32_4;

architecture imp of rom_32_4 is
begin
  process (inputs...) -- Outputs and next state function
  begin
    if (reset = '1') then
      for i in 0 to 6 loop -- unrolled at compile time
        tmp(i+1) <= tmp(i);
      end loop;
    else
      tmp(0) <= SI;
    end if;
  end process;
end imp;
```

The VHDL Hardware Description Language – p. 23/

**Rocket Science: FSMs**

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

entity myFSM is
  port( ... );
end myFSM;

architecture impl of myFSM is
begin
  process (inputs...) -- Outputs and next state function
  begin
    if (Clk'event and Clk = '1') then
      if (en = '1') then
        next_state <= STATE2;
      else
        next_state <= STATE1;
      end if;
    end if;
  end process;
end imp;
```

The VHDL Hardware Description Language – p. 25/

**The Traffic Light Controller**

This controls a traffic light at the intersection of a busy highway and a farm road. Normally, the highway light is green but if a sensor detects a car on the farm road, the highway light turns yellow then red. The farm road light then turns green until there are no cars or a long time. Then, the farm road light turns yellow then red, and the highway light returns to green. The inputs to the machine are the car sensor, a short timeout signal, and a long timeout signal. The outputs are a timer start signal and the colors of the highway and farm road lights.

architecture imp of tlc is
signal current_state, next_state : std_ulogic_vector(1 downto 0);
constant HG : std_ulogic_vector := "00";
constant HY : std_ulogic_vector := "01";
constant FY : std_ulogic_vector := "10";
constant FG : std_ulogic_vector := "11";
begin

P1: process (clk) -- Sequential process
begin
if (clk'event and clk = '1') then
  current_state <= next_state;
end if;
end process P1;

-- Combinational process
-- Sensitive to input changes, not clock
P2: process (current_state, reset, cars, short, long)
begin
if (reset = '1') then
  next_state <= HG;
  start_timer <= '1';
else
  case current_state is
  when HG =>
    highway_yellow <= '0';
    highway_red <= '0';
    farm_yellow <= '0';
    farm_red <= '1';
    if (cars = '1' and long = '1') then
      next_state <= HY;
      start_timer <= '1';
    else
      next_state <= HG;
      start_timer <= '0';
    end if;
  when HY =>
    highway_yellow <= '1';
    highway_red <= '0';
    farm_yellow <= '0';
    farm_red <= '1';
    if (short = '1') then
      next_state <= FG;
      start_timer <= '1';
    else
      next_state <= HY;
      start_timer <= '0';
    end if;
  when FG =>
    highway_yellow <= '0';
    highway_red <= '1';
    farm_yellow <= '0';
    farm_red <= '0';
    if (cars = '0' or long = '1') then
      next_state <= FY;
      start_timer <= '1';
    else
      next_state <= FG;
      start_timer <= '0';
    end if;
  when FY =>
    highway_yellow <= '0';
    highway_red <= '1';
    farm_yellow <= '1';
    farm_red <= '0';
    if (short = '1') then
      next_state <= HG;
      start_timer <= '1';
    else
      next_state <= FY;
      start_timer <= '0';
    end if;
  when others =>
    next_state <= "XX";
    start_timer <= 'X';
    highway_yellow <= 'X';
    highway_red <= 'X';
    farm_yellow <= 'X';
    farm_red <= 'X';
  end case;
end if;
end process P2;
end imp;