The VHDL Hardware Description Language

CSEE W4840

Prof. Stephen A. Edwards

Columbia University

Why HDLs?

1970s: SPICE transistor-level netlists

library ieee;
use ieee.std_logic_1164.all;
entity ALU is
  port(A: in std_logic_vector(1 downto 0);
       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;

Why HDLs?

1980s: Graphical schematic capture programs

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity multlexer_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 multlexer_4_1;

architecture imp of multlexer_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
        "XXXXXXXXXXXXXXXXX";
end imp;

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

Signal

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

Dataflow Expression

VHDL: Two-bit Counter

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 => C(2));
end imp;

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_logic;
       sum, carry : out std_logic);
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); -- combinational logic
end imp;

VHDL: Two-bit Counter

library ieee;
use ieee.std_logic_1164.all;
entity multlexer_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 multlexer_4_1;

architecture imp of multlexer_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
        "XXXXXXXXXXXXXXXXX";
end imp;

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

library ieee;
use ieee.std_logic_1164.all;
entity multiplexer_4_1 is
  port(i0, i1 : 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 <= i0 when (s0 = '0' and s1 = '0') else
       i1 when (s0 = '1' and s1 = '0') else
       i2 when (s0 = '0' and s1 = '1') else
       i3 when (s0 = '1' and s1 = '1') else
        "XXXXXXXXXXXXXXXXX";
end imp;
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 usewith of multiplexer_4_1 is
signal sels : std_logic_vector(1 downto 0);
begnin
sels <= s1 & s0; -- Vector concatenation
with sels select
z <=
in0 when "00",
in1 when "01",
in2 when "10",
in3 when "11",
"XXXXXXXXXXXXXXXX" when others;
end usewith;

library ieee;
use ieee.std_logic_1164.all;

entity dec1_8 is
port (sel : in std_logic_vector(2 downto 0);
res : out std_logic_vector(7 downto 0));
end dec1_8;

architecture imp of dec1_8 is
begin
res <= "00000001" when sel = "000" else
"00000010" when sel = "001" else
"00000100" when sel = "010" else
"00001000" when sel = "011" else
"00010000" when sel = "100" else
"00100000" when sel = "101" else
"01000000" when sel = "110" else
"10000000";
end imp;

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

entity adder is
port (A, B : in std_logic_vector(7 downto 0);
CI : in std_logic;
SUM : out std_logic_vector(7 downto 0);
CO : out std_logic);
end adder;

architecture imp of adder is
signal tmp : std_logic_vector(8 downto 0);
begnin
tmp <= conv_std_logic_vector((conv_integer(A) +
conv_integer(B) +
conv_integer(CI)), 9);
SUM <= tmp(7 downto 0);
CO <= tmp(8);
end imp;

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

entity alu is
port (A, B : in std_logic_vector(7 downto 0);
ADD : in std_logic;
RES : out std_logic_vector(7 downto 0));
end alu;

architecture imp of alu is
begin
RES <= A + B when ADD = '1' else
A - B;
end imp;

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

entity comparator is
port (A, B : in std_logic_vector(7 downto 0);
GE : out std_logic);
end comparator;

architecture imp of comparator is
begin
GE <= '1' when A >= B else '0';
end imp;

library ieee;
use ieee.std_logic_1164.all;

entity rippleadder is
port (a, b : in std_logic_vector(3 downto 0);
cin : in std_logic;
sum : out std_logic_vector(3 downto 0);
cout : out std_logic);
end rippleadder;

architecture imp of rippleadder is
signal c : std_logic_vector(4 downto 0);
begnin
c(0) <= cin;
G1: for m in 0 to 3 generate -- at compile time
sum(m) <= a(m) xor b(m) xor c(m);
c(m+1) <= (a(m) and b(m)) or (b(m) and c(m)) or
(a(m) and c(m));
end generate G1;
cout <= c(4);
end imp;

library ieee;
use ieee.std_logic_1164.all;

entity flipflop is
port (Clk, D : in std_logic;
Q : out std_logic);
end flipflop;

architecture imp of flipflop is
begin
process (Clk)
begin
if (Clk'event and Clk = '1') then
Q <= D;
end if;
end process P1;
end imp;

library ieee;
use ieee.std_logic_1164.all;

entity flipflop_reset is
port (Clk, Reset, D : in std_logic;
Q : out std_logic);
end flipflop_reset;

architecture imp of flipflop_reset is
begin
P1: process (Clk)
begin
if (Clk'event and Clk = '1') then
if (Reset = '1') then Q <= '0';
else Q <= D;
end if;
end process P1;
end imp;
**Four-bit binary counter**

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity counter is
  port(
    Clk : in std_logic;
    Reset : in std_logic;
    Q : out std_logic_vector(3 downto 0));
  end counter;
architecture impl of counter is
begin
  process (Clk)
  begin
    if (Clk'event and Clk = '1') then
      if (Reset = '1') then
        Count <= '0000';
      else
        Count <= Count + 1;
      end if;
    end process;
    Q <= Count; -- copy count to output
  end imp;
end counter;
```

**Eight-bit serial in/out shift register**

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity shifter is
  port(
    Clk : in std_logic;
    SI : in std_logic;
    SO : out std_logic_vector(7 downto 0));
  end shifter;
architecture impl of shifter is
begin
  process (Clk)
  begin
    if (Clk'event and Clk = '1') then
      tmp(0) <= SI;
    end if;
    if (we = '1') then
      tmp(i+1) <= tmp(i);
    else
      tmp(i+1) <= tmp(i);
    end if;
    end process;
    do <= tmp(7); -- Copy to output
  end imp;
end shifter;
```

**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(3 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
  signal count : std_logic_vector(3 downto 0);
  signal RAM : ram_type;
  process (Clk)
  begin
    if (Clk'event and Clk = '1') then
      if (Reset = '1') then
        count <= count + 1;
      else
        count <= count + 1;
      end if;
    end if;
    end process;
    Q <= count; -- copy count to output
  end imp;
end ram_32_4;
```

**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 after a long timeout. 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.


**Rocket Science: FSMs**

**Structure of FSMs in VHDL**

```vhdl
entity myFSM is
  port( ... );
end myFSM;
architecture imp of myFSM is
begin
  process (clk) -- State holding element process
  begin
    if (clk'event and clk = '1') then
      current_state <= next_state;
    end if;
  end process;
  end imp;
end myFSM;
```

**FSM for the Traffic Light Controller**

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity tlc is
  port (;
    clk : in std_ulogic;
    reset : in std_ulogic;
    cars : in std_ulogic;
    short : in std_ulogic;
    long : in std_logic;
    highway_yellow : out std_ulogic;
    highway_red : out std_ulogic;
    farm_yellow : out std_ulogic;
    farm_red : out std_ulogic;
    start_timer : out std_ulogic;
    end tlc;
end tlc;
```

**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(3 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
  signal count : std_logic_vector(3 downto 0);
  signal RAM : ram_type;
  process (Clk)
  begin
    if (Clk'event and Clk = '1') then
      if (Reset = '1') then
        count <= count + 1;
      else
        count <= count + 1;
      end if;
    end if;
    end process;
    Q <= count; -- copy count to output
  end imp;
end ram_32_4;
```

**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 after a long timeout. 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
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;