Chapter 2
Hardware Description Languages

Dr. Curt Nelson

Engr433 – Digital Design

Outline

Overview of hardware description languages;
Basic VHDL concepts via an example;
VHDL in development flow.

Programming Languages

- Can we use C or Java as a High-Level Description Language (HDL) targeting hardware?
- A computer programming language
  - Semantics ("meaning");
  - Syntax ("grammar").
- Development of a language
  - Study the characteristics of the underlying processes;
  - Develop syntactic constructs and their associated semantics to model and express these characteristics.
- Traditional programing languages are modeled after a sequential process
  - Operations performed in a sequential order;
  - Help human's thinking process to develop an algorithm step by step.

HDL

- Characteristics of digital hardware
  - Connections of parts;
  - Concurrent operations;
  - Concept of propagation delay and timing.
- Characteristics cannot be captured by traditional Programming Languages.
- Therefore, we require a **Hardware Specific Language**
  - VHDL
  - Verilog

Uses of an HDL Program

- Formal documentation;
- Generate input to a simulator;
- Generate input to a synthesizer.

HDL Features

- Encapsulate the concepts of entity, connectivity, concurrency, and timing;
- Incorporate propagation delay and timing information;
- Consist of constructs for structural implementation;
- Incorporate constructs for behavioral descriptions (sequential execution of a traditional PL);
- Describe the operations and structures at the gate-level and register-transfer level;
- Consist of constructs to support hierarchical design processes.

Two HDLs Used Today

- VHDL and Verilog;
- Syntax and "appearance'" of the two languages are very different;
- Capabilities and scopes are quite similar;
- Both are industrial standards and are supported by most software tools.

VHDL

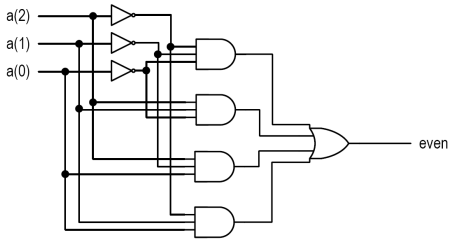
- VHDL - VHSIC (Very High Speed Integrated Circuit) HDL;
- Initially sponsored by DoD as a hardware documentation standard in early 1980's;
- Transferred to IEEE and ratified as IEEE standard 1176 in 1987 (known as VHDL-87);
- Major modification in 1993 (known as VHDL-93);
- Revised continuously.

IEEE Extensions

- IEEE standard 1076.1 Analog and Mixed Signal Extensions (VHDL-AMS);
- IEEE standard 1076.2 Mathematical Packages;
- IEEE standard 1076.3 Synthesis Packages;
- IEEE standard 1076.4 VHDL Initiative Towards ASIC Libraries (VITAL);
- IEEE standard 1076.6 VHDL Register-Transfer-Level (RTL) Synthesis;
- IEEE standard 1164 Multi-value Logic System for VHDL Model Interoperability;
- IEEE standard 1029 VHDL Waveform and Vector Exchange to Support Design and Test Verification (WAVES).

Example - Even Parity Detection Circuit

- Inputs: a(2), a(1), a(0)
- Output: even



a(2)	a(1)	a(0)	even
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

$$even = a(2)' \cdot a(1)' \cdot a(0)' + a(2)' \cdot a(1) \cdot a(0) + a(2) \cdot a(1)' \cdot a(0) + a(2) \cdot a(1) \cdot a(0)'$$

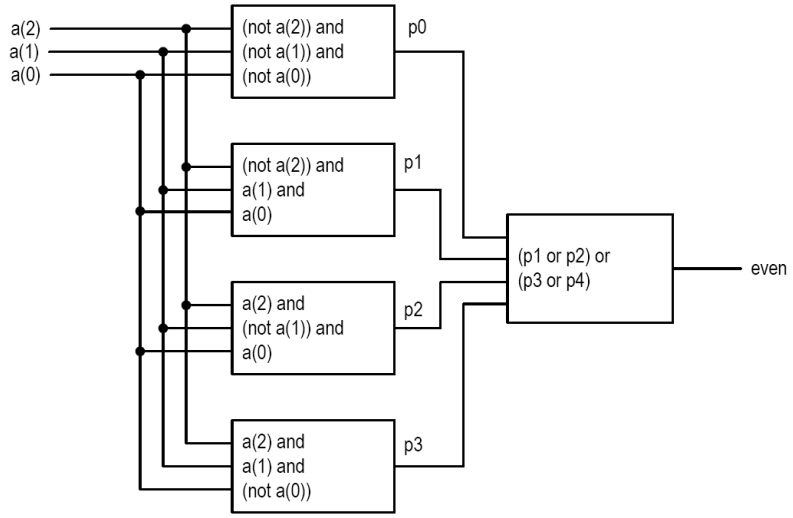
VHDL Listing 2.1 – Even Parity Detector

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

-- entity declaration
entity even_detector is
   port(
      a: in std_logic_vector(2 downto 0);
      even: out std_logic
   );
end even_detector;

-- architecture body
architecture sop_arch of even_detector is
   signal p1, p2, p3, p4 : std_logic;
begin
   even <= (p1 or p2) or (p3 or p4) after 20 ns;
   p1 <= (not a(2)) and (not a(1)) and (not a(0)) after 15 ns;
   p2 <= (not a(2)) and a(1) and a(0) after 12 ns;
   p3 <= a(2) and (not a(1)) and a(0) after 12 ns;
   p4 <= a(2) and a(1) and (not a(0)) after 12 ns;
end sop_arch ;
```

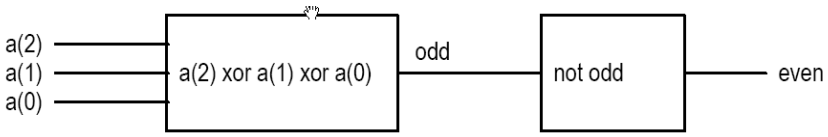
Program Interpretation

- Entity declaration
  - i/o ports (the top-level inputs and outputs)
- Architecture body
  - Signal declaration (internal signals)
  - Each concurrent statement
    - Can be thought of as a circuit part;
    - Contains timing information.
  - Architecture body can be thought of as a *collection of parts*

Conceptual Interpretation



VHDL Listing 2.2 – Using XOR Network



- Same entity declaration;
- Implicit delay assumed (**very** small number) when delay is not specified

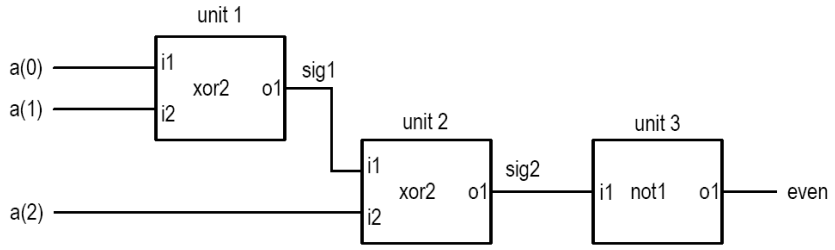
```
architecture xor_arch of even_detector is
   signal odd: std_logic;
begin
   even <= not odd;
   odd <= a(2) xor a(1) xor a(0);
end xor_arch;
```

Structural Description

- In structural view, a circuit is constructed by smaller parts;
- Structural description specifies the types of parts and connections;
- Essentially a textual description of a schematic;
- Done by using the keyword **component** in VHDL
  - First *declared* (make known);
  - Then *instantiated* (used).

Structural Example

- Even detector using previously designed components (xor2 and not1).



VHDL Listing 2.3 – Structural Example

```
architecture str_arch of even_detector is
  -- declaration for xor gate
  component xor2
    port(
      i1, i2: in std_logic;
      o1: out std_logic
    );
  end component;
  -- declaration for inverter
  component not1
    port(
      i1: in std_logic;
      o1: out std_logic
    );
  end component;
  signal sig1,sig2: std_logic;
begin
  -- instantiation of the 1st xor instance
  unit1: xor2
    port map (i1 => a(0), i2 => a(1), o1 =>
sig1);
  -- instantiation of the 2nd xor instance
  unit2: xor2
    port map (i1 => a(2), i2 => sig1, o1 =>
sig2);
  -- instantiation of invertor
  unit3: not1
    port map (i1 => sig2, o1 => even);
end str_arch;
```

VHDL Listing 2.4 - Behavioral Description

```
library ieee;
use ieee.std_logic_1164.all;
entity xor2 is
   port(
      i1, i2: in std_logic;
      o1: out std_logic);
end xor2;

architecture beh_arch of xor2 is
begin
   o1 <= i1 xor i2;
end beh_arch;

library ieee;
use ieee.std_logic_1164.all;
entity not1 is
   port(
      i1: in std_logic;
      o1: out std_logic);
end not1;
architecture beh_arch of not1 is
begin
   o1 <= not i1;
end beh_arch;
```

"Behavioral" Description

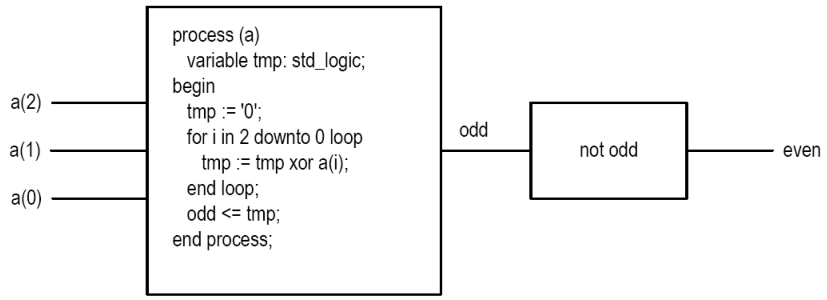
- No formal definition on "behavioral" in VHDL;
- VHDL "process" - a language construct to encapsulate **sequential semantics**
  - The entire process is a concurrent statement;
  - Syntax:

```
process (sensitivity_list)
   variable declaration;
begin
   sequential statements;
end process;
```

Listing 2.5

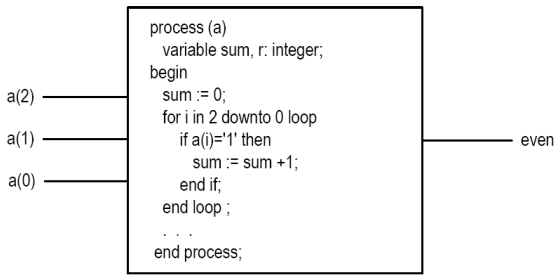
```
architecture beh1_arch of even_detector is
signal odd: std_logic;
begin
   even <= not odd;
   process (a)
      variable tmp: std_logic;
   begin
      tmp := '0';
      for i in 2 downto 0 loop
         tmp := tmp xor a(i);
      end loop;
      odd <= tmp;
   end process;
end beh1_arch;
```

Conceptual Interpretation



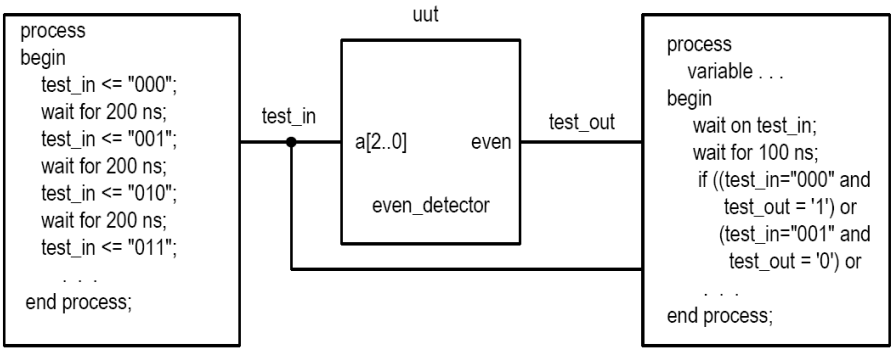
Listing 2.6 – Another Behavioral Description

```
architecture beh2_arch of even_detector is
begin
   process (a)
      variable sum, r: integer;
   begin
      sum := 0;
      for i in 2 downto 0 loop
          if a(i)='1' then
             sum := sum +1;
          end if;
      end loop ;
      r := sum mod 2;
      if (r=0) then
         even <= '1';
      else
          even <='0';
      end if;
   end process;
```



Testbench

- A "virtual" experiment table
  - Circuit to be tested;
  - Input stimuli (e.g., function generator);
  - Output monitor (e.g., logic analyzer).



VHDL Listing 2.7 – Simple Testbench

```
library ieee;
use ieee.std_logic_1164.all;
entity even_detector_testbench is
end even_detector_testbench;

architecture tb_arch of even_detector_testbench is
   component even_detector
      port(
         a: in std_logic_vector(2 downto 0);
         even: out std_logic);
   end component;
   signal test_in: std_logic_vector(2 downto 0);
   signal test_out: std_logic;

begin
   -- instantiate the circuit under test
   uut: even_detector
      port map( a=>test_in, even=>test_out);
```

Testbench - Continued

```
-- test vector generator
process
begin
   test_in <= "000";
   wait for 200 ns;
   test_in <= "001";
   wait for 200 ns;
   test_in <= "010";
   wait for 200 ns;
   test_in <= "011";
   wait for 200 ns;
   test_in <= "100";
   wait for 200 ns;
   test_in <= "101";
   wait for 200 ns;
   test_in <= "110";
   wait for 200 ns;
   test_in <= "111";
   wait for 200 ns;
end process;
```

Testbench - Continued

```
-- verifier
process
   variable error_status: boolean;
begin
   wait on test_in;
   wait for 100 ns;
   if ((test_in="000" and test_out = '1') or
       (test_in="001" and test_out = '0') or
       (test_in="010" and test_out = '0') or
       (test_in="011" and test_out = '1') or
       (test_in="100" and test_out = '0') or
       (test_in="101" and test_out = '1') or
       (test_in="110" and test_out = '1') or
       (test_in="111" and test_out = '0'))
   then
      error_status := false;
   else
      error_status := true;
   end if;
   -- error reporting
   assert not error_status
      report "test failed."
      severity note;
end tb_arch;
```

Configuration

- Multiple architecture bodies can be associated with an entity declaration
  - Like IC chips and sockets.
- VHDL configuration specifies the *binding*

```
configuration demo_config of even_detector_testbench is
   for tb_arch
      for uut: even_detector
         use entity work.even_detector(sop_arch);
      end for;
   end for;
end demo_config;
```

Synthesis

- Design realized by available hardware components;
- Many VHDL constructs can be synthesized (e,g, file operation, floating-point data type, division)
  - Only small subset can be used;
  - E.g., some control structures like loops or conditional branches can simulate easily but cannot be efficiently mapped to hardware.
- Syntactically correct code ≠ Synthesizable code;
- Synthesizable code ≠ Efficient code;
- Synthesis software only performs transformation from a text file to available hardware.

Summary

- This course focuses on hardware (the **H** not the **L** in VHDL)
- Emphasis on coding for synthesis
  - Code accurately describing the underlying hardware structure;
  - Code providing adequate info to guide synthesis software to generate efficient implementations.