Lecture 2
Hardware Description Language (HDL): VHSIC HDL (VHDL)

Pinit Kumhom

VLSI Laboratory
Dept. of Electronic and Telecommunication Engineering (KMUTT)
Faculty of Engineering
King Mongkut’s University of Technology Thonburi

pinit.kum@kmutt.ac.th
A Brief History of HDLs

• 30 yrs ago – schematic using a ruler and a pencil
• 1980s
  – Schematic using schematic editor tools
  – Beginning of hardware description language (HDL)
• 1990s
  – Usages of HDL accelerated as PLDs, CPLDs, FPGAs become inexpensive and commonplace
  – ASIC densities ↑ => difficult for using schematics alone => HDL
    • HDL for both top-level and module-level design of PLD, CPLD, FPGA or ASIC.
    • Schematics for board-level interconnections among these devices and other components such as memories, microprocessors, and SSI/MSI “glue” logic.
• PALASM (PAL Assembler) – the first commercial HDL
  – By Monolithic Memories, Inc. in 1980s
  – Assembly-like language to provide a text-based means to specify the info. to be programmed to a Programmable Array Logic (PAL)
• Improved PALASM, CUPL, and ABEL – the 2nd generation HDL
  – Logic minimization
  – “High-level” statement construct such as “if-then-else”, and “case”
• VHDL and Verilog
  – Started out as “simulation” languages before being used as languages for describing “synthesizable” hardware.
Why HDL?

• Two reasons that change the way typical digital design is done.
  – Fast growing of chip manufacturing technology -> complex digital systems
    • Programmable VLSI, FPGA, CPLD
    • ASIC
  – Availability of synthesizable HDL

• HDL works in a similar way a high-level programming language (C/C++, Java, etc) does in software development
  – Communicate between designers and EDA tools, among designers, and designer teams and other teams (to some extend)
  – Synthesized hardware not necessary the best, but delivered in time (or just in time)
  – Needs HDL tool suites
HDL Tool Suites

• Necessary tools
  – A text editor
  – The compiler
  – A synthesizer (or synthesis tool)
  – A simulator

• Useful programs and utilities
  – A template generator
  – A schematic viewer
  – A translator
  – A timing analyzer
  – A back annotator
HDL Tool Suites: A Text Editors and The Compiler

• A text editor
  – A design entry tool
  – Often contains HDL-specific features
    • Recognizing specific filename extensions
    • Recognizing HDL reserved words and display them with different colors

• The compiler
  – Parsing the HDL program
  – Check its syntax -> report any syntax errors
  – Figure out what the program really “says”
  – Give a result in an intermediate, technology-neutral form
    • Unambiguous description of the interconnection and logic operations specified by the HDL program
    • Not quite a hardware realization
HDL Tool Suites: Synthesizer and Simulator

- **A synthesizer** (or **synthesis tool**)
  - Target the design to a **specific hardware technology**, such as PLD, CPLD, FPGA, or ASICs.
  - Need one or more **libraries**
    - having specifics on the targeted technology, such as the gates and flip-flops
    - May contain larger-scale components such as multi-bit adders, registers, and counters

- **A simulator**
  - Two inputs
    - the HDL program representing the designed hardware
    - A time sequence of inputs generated by
      - a test bench
      - waveform editor
  - Run the specific input sequence on the described hardware and
  - Determines the values of the hardware internal signals and its output over a specified period of time.
HDL Tool Suites: Helpful programs and utilities

- A template generator
  - create a text file with the outline of a commonly used program structure
- A schematic viewer
  - May create a schematic diagram corresponding to an HDL program
- A translator
  - Translate the compiler’s intermediate-language output to a real device, such as PLD, FPGA, or ASIC.
  - May integrate associate fitter and chip viewer.
- A timing analyzer
  - Calculates the delays through some or all of the signal paths in the final chip
  - Produces a report showing the worst-case paths
- A back annotation
  - Insert delay causes or statements in the original HDL source program corresponding to the delays calculated by the timing analyzer
  - Allow subsequent simulators to include timing, whether the source program is simulated.
The VHDL: The beginning

- Started out as a project supported by the US Department of Defense (DoD) and IEEE (Institute of Electrical and Electronic Engineering) in mid 1980’s

- The early features (still are)
  - Design may be decomposed hierarchically
  - Each design element
    - a well-defined interface (for connecting to others)
    - a precise functional specification (for simulating it)
  - A functional specification can use either
    - a behavioral algorithm
    - an actual hardware structure
  - Concurrency, timing, and clocking can all be modeled.
    - Handle both asynchronous and synchronous hardware
  - The logical operation and timing behavior can be simulated.

A documentation and modeling language, allowing the behavior of digital-system design to be precisely specified and simulated.
The VHDL: A quantum leap and the standards

• VHDL + its simulation environment were important innovation by themselves.

• A quantum leap with commercial development of VHDL synthesis tools
  – Create logic-circuit structure from VHDL behavior descriptions

• VHDL standards
  – was first standardized in 1987 (VHDL-1987)
  – Extended in 1993 (VHDL-1993)
  – Extended again in 2002 (VHDL-2002)
Basic VHDL Concept Via an Example

- Problem: want a circuit for detecting even parity of a 3-bit input signals
- Based on the digital design principles and problem solving skills we get...

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

Truth table – a way to describe circuit’s function

<table>
<thead>
<tr>
<th>a(2)</th>
<th>a(1)</th>
<th>a(0)</th>
<th>even</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
First VHDL description of the even detection circuit

```vhdl
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 ;
```
Basic VHDL Concept Via an Example

First VHDL description of the even detection circuit

```vhdl
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
  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 15 ns;
end sop_arch;
```

Learning VHDL Concept:

0. About the notations:
   - Words in **bold face** s.a. library, `begin`, `entity` → reserved words
   - Words behind `'--'` in each line → comments
     
     e.g. **-- this is a comment**

1. The VHDL **structure**: 2 major units
   - **Entity declaration**
   - **Architecture body**
Basic VHDL Concept Via an Example

First VHDL description of the even detection circuit

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

Learning VHDL Concept:

Entity declaration

- specifying the circuit’s input and output **ports**
  - Input ports: `a` represent `a(2), a(1), a(0)`
  - Output port: `even`
- It provides a way for connecting the circuit to the outside, or the interface
Basic VHDL Concept Via an Example

First VHDL description of the even detection circuit

```vhdl
architecture sop_arch of even_detector is
begin
    signal p1, p2, p3, p4 : std_logic;
    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;
```

Learning VHDL Concept:

**Architecture body**

- specifying the circuit’s *internal operation* or *organization*
- The fundamental building block inside the architecture body is the *concurrent statement*.

The circuit

Area for declarations s.a. signal declaration

Description of circuit’s operation or organization

The outside
First VHDL description of the even detection circuit

\[ \text{even} := (p1 \text{ or } p2) \text{ or } (p3 \text{ or } p4) \text{ after } 20 \text{ ns}; \]

Learning VHDL Concept:
Concurrent statement

- A circuit part
  - Left-hand-side = an output signal
  - Right-hand-side
    - All signals = input signals
    - Expression = operation performed by the circuit part
    - The **after** cause incorporates the propagation delay

The result of the operation is available at the output **after** a specific amount of a propagation delay.

The circuit

The outside
Basic VHDL Concept Via an Example

First VHDL description of the even detection circuit

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;

Linked by common signals or the nets

Conceptual diagram of the sop_arch
Basic VHDL Concept Via an Example

First VHDL description of the even detection circuit

```vhdl
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;
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;
p4 <= a(2) and a(1) and (not a(0)) after 12 ns;
```

Learning VHDL Concept:

Concurrent statement

- **Order** of the concurrent statements **does not matter**
- Concurrent statements are **independent** and can be activated in parallel
- Whenever an **input** of concurrent statement **changes**, it is “awakened” and **evaluate** the operations accordingly.
Basic VHDL Concept Via an Example

1. VHDL description of the even detection circuit – dataflow description

```vhdl
even <= (p1 or p2) or (p3 or p4) after 20 ns;
```

Learning VHDL Concept: **Concurrent statement**

- The **incorporation of propagation delay** is the key ingredient in…
  - modeling the operation of hardware, and
  - insuring the proper interpretation of VHDL code.

- When the after cause is omitted = an implicit \( \delta \)-delay (delta delay) is incorporated.

- \( \delta \)-delay = infinitesimal delay (more than zero but smaller than any physical number.)

- **There is always a propagation delay associated with a concurrent statement regardless of appearance of the after cause.**
1.1 Other ways of describing the even detection circuit

From the problem statement directly to the concurrent statement (without using the truth table)

- Even parity = the number of ‘1’ in the input signals is even (0 (=none), 2, 4, ..)
- The **xor** operation is ‘1’ whenever the number of ‘1’ in its input is odd. (1,3,5,...)
- **even parity = not of the xor of all input signals**

\[
even = (a(2) \oplus a(1) \oplus a(0))'
\]
Basic VHDL Concept Via an Example

Ways of describing (the even detection) circuit

2. Structural description

• For describing the structural view
• A circuit is constructed from smaller parts.
• Structural description specifies:
  – what types of parts are used, and
  – how these parts are connected.
• Treating a concurrent statement as a circuit part = our interpretation (not consider a real structural description)
• Formal VHDL structural description uses the concept of component
  – A component can be either an existing or a hypothetical part.
  – It must first declared (make known), then
  – can be instantiated (actually used) in the architecture body as needed.
Basic VHDL Concept Via an Example

Ways of describing (the even detection) circuit

2. Structural description (structural view)

Component declaration

```vhdl
component xor2
  port ( i1, i2: in std_logic;
         o1: out std_logic
    );
end component;
```

Component instantiation

```vhdl
architecture str_arch of even_detector is

  component xor2
    port(
      i1, i2: in std_logic;
      o1: out std_logic
    );
  end component;

  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 inverter
  unit3: not1
  port map (i1 => sig2, o1 => even);

end str_arch;
```

The connection is done via the same signal name.
Basic VHDL Concept Via an Example

Ways of describing (the even detection) circuit

2. Structural description (structural view)

• The uses of structural description
  – Facilitate the *hierarchical design*
  – Provide a method to use predesigned circuits
    • IP cores
    • Library cells from device vendors (Xilinx, Altera..)
  – To represent the result from a synthesis— a gate or cell-level *netlist*. 
Ways of describing (the even detection) circuit

2. Structural description (structural view)

Component declaration

```vhdl
component xor2
    port (  
        i1, i2: in std_logic;  
        o1: out std_logic
    );
end component;
```
Ways of describing (the even detection) circuit

3. Abstract behavioral description

- In a Large design -> **just** want to study **system operation** first because
  - implementation can be very complex, and
  - construction is very time-consuming

- **Human reasoning and algorithms** resemble a **sequential process** => 
  sequential semantics of traditional language is more adequate.

- **VHDL** provides language constructs that resemble the **sequential semantics** including
  - the **use of variables**, and
  - sequential execution

- Encapsulated in a special language construct known as a **process**.

- This kind of codes sometimes known as **behavioral description** \(\rightarrow\) this is **not precise definition**

- All codes except for pure component instantiation \(\rightarrow\) behavioral
Basic VHDL Concept Via an Example

Ways of describing (the even detection) circuit

3. Abstract behavioral description

The basic skeleton of a process

- sensitivity list -- a set of signals
  - When a signal in the list changes, the process is activated.

- Inside the process the semantics is similar to that of a traditional programming
  - variables can be used, and
  - sequential execution

```vhdl
process(sensitivity_list)
variable declaration;
begin
  sequential statements;
end process;
```
Ways of describing (the even detection) circuit

3. Abstract behavioral description

```
architecture beh1_arch of even_detector is
  signal odd: std_logic;
begin
  -- inverter
  even <= not odd;
  xor network for odd parity
  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;
```

A process is treated as **one individual part** whose behavior is specified by the sequential statements.

Conceptual diagram of the beh1_arch
Ways of describing (the even detection) circuit

3. Abstract behavioral description

```vhdl
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;
end beh2_arch;
```

While the code is very straightforward and easy to understand, it provides no clues about the underlying structure or how to realize the code in hardware.
Basic VHDL Concept Via an Example

VHDL Simulation and Testbench

- One major use of a VHDL code (program) is the simulation
  - To study the operation of the circuits, or
  - To verify the correctness of the design

- Performing simulation is similar to doing experiment with physical a physical circuits

Doing experiment

- A stimulus e.g. signal generators
- Output observer e.g. Logic analyzer
- A physical Circuit
- Circuit’s input
- Circuit’s output

Performing simulation

- testbench
  - A stimulus generator or test vector generator
  - VHDL utility routines
  - Wave editors

- Output observer
  - VHDL utility routines
  - Human
- A model of the circuit (e.g. VHDL code)
- Circuit’s input
```vhdl
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);
```
Basic VHDL Concept Via an Example

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

```vhdl
-- 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;
```

```vhdl
-- verifier
process
begin
  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 process;
end tb_arch;
```
Basic VHDL Concept Via an Example

Configuration – a design unit for binding entities with architecture bodies

- VHDL intentionally separates the entity declaration and the architecture body into 2 design units.
  - One entity can be associated with multiple architecture bodies = a circuit can be viewed and described in many different ways.
- At a time of simulation, we can choose a specific architecture body to bind with the entity.

![Diagram showing entity and architecture bodies]

VHDL code associates even_detector entity to 5 architecture bodies.

At a time of simulation, we bind the sop_arch architecture body to the even_detector.
Basic VHDL Concept Via an Example

Configuration – a **design unit** for binding entities with architecture bodies

- Example in the testbench where the `even_detector` is instantiated

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

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;
```

- A configuration is not always needed.
- When an instantiation is not clearly configured, the entity is automatically bound with the last compiled architecture body.
VHDL in Development Flow

• Choices of the constructs in the example is not accidental
• They are carefully selected to
  – facilitate the development flow and
  – the different between
    • coding for modeling and
    • coding for synthesis
VHDL in Development Flow

### Scope of VHDL

- Start with developing abstract behavioral description (s.a. beh2_arch in the example) and a testbench that later can be used to verify the synthesis results.

  - In a very large system, the abstract behavioral descriptions can be synthesized to a very complex hardware (don’t want) or cannot synthesized at all.

- Once the specification and the behavior of the system is understood, the RTL descriptions (s.a. the xor_arch and beh1_arch) are developed.

  - provide a “sketch” of the underlying hardware structure.
VHDL in Development Flow

Coding for synthesis
• After developed, a VHDL can be executed in a simulator or synthesizer

In Simulation
• A design is realized in virtual environment: the software simulator
  – Use host computer instructions to mimic operation of the circuit
  – Done sequentially (multiple constructs and operators of VHDL share the same resource in time multiplexing fashion)
  – All VHDL constructs can be simulated

In synthesis
• All VHDL constructs and operators of the VHDL are mapped to hardware
• Only a subset of VHDL can be used.
• Example of VHDL constructs that cannot be synthesized
  – File operations, assertion statements
  – Floating point multiplier – too complex to synthesize
• IEEE define a subset of VHDL that is suitable for synthesis in IEEE standard 1076.6
  – It’s restricted but rich and flexible
  – No guarantee about the efficiency of the synthesized circuit
  – Depend heavily on the initial description