Introduction to A Hardware Description Language – VHDL

Outline

- HDL and VHDL Overview
- VHDL Language Organization
- Basic VHDL Modeling
  - Entity declaration
  - Architecture declaration
  - Data objects
  - Component
HDL-Overview

- What is Hardware?

- Hardware Description Language (HDL) = "Programming" language for modeling of (digital) hardware.

- HDL: VHDL (very-high-speed integrated circuits (VHSIC)), Verilog

Application of HDL

- HDL offer design reuse capabilities.
  - The corresponding HDL model can be re-used in several designs/projects.
  - Frequently needed function blocks (macros) are collected in model libraries.
Range of Use

**Behavioural VHDL**
- **Specification**
- **System Design**
- **Validation**

**RTL VHDL**
- **Logic Design**
- **Validation**

**Netlist VHDL**
- **Circuit Design**
- **Validation**
- **Layout**
- **Validation**

**Graphical Textual Tables**
- Specification: Description of the system requirements
- System Design: Modelling the behaviour
- Logic Design: Modelling the structure
- Circuit Design: Automatic conversion of structural description
- Validation: Check function through simulation
- Provide input stimuli
- Check expected response

**VHDL – Overview**

- Very-High-Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL):
  - Modeling of digital systems
  - **Concurrent and sequential** statements
  - Machine-readable specification
  - Man- and machine-readable documentation

- International Standards:
VHDL Information

- Reference Books for VHDL:
  - J. Bhasker "A VHDL Primer (3rd Edition)", Prentice Hall
  - Yalamanchili "Introductory VHDL from Simulation to Synthesis"
  - D. L. Perry “VHDL Programming by Example” McGrawHill

- Useful web site for VHDL:
  - http://www.eda.org/rassp/vhdl

Concepts of VHDL

- Execution of assignments:
  - Sequential:
    - Executed one after another, like in software programming languages.
    - Can override the effects of previous statements.
  - Concurrent:
    - Active continuously.
    - The order of statements is not relevant.
    - Suited to model the parallelism of hardware.

- Abstraction: description of different parts of a system.
  - On every abstraction level, only the essential information is considered, nonessential information is left out.
Abstraction levels in Digital Design

- **Behavioral level:**
  - Functional description of the model is outlined.
  - No system clock, signal transitions are asynchronous with respect to the switching time.
  - Simulation only, but *typically not synthesizable.*

- **Register level (RTL):**
  - The design is divided into combinational logic and storage elements.
  - Storage elements (Flip-Flops, latches) are controlled by a system clock.
  - Synthesizable.

- **Logic level:**
  - The design is represented as a netlist with logic gates (AND, OR, NOT,...) and storage elements.

- **Layout level:**
  - The different cells of the target technology are placed on the chip and the connections are routed.
  - After the layout has been verified, the circuit is ready for the production process (submitted to manufacturing factory for chip fabrication).
Information Content of Abstraction Levels

Abstraction levels and VHDL

Abstraction Levels
- Behaviour
- RTL
- Logic
- Layout

Automation Tools
- Behavioural Synthesis
- Logic Synthesis
- Placement and Route
**Behavioral Description in VHDL**

The function can be modeled as a simple equation (e.g. \( i_1 + i_2 \times i_3 \)) plus a delay of 100 ns.

\[
\text{o} \leftarrow \text{transport} \ i_1 + i_2 \times i_3 \ \text{after} \ 100 \ \text{ns};
\]

**Behavioral Synthesis**

- Only very simple behavioral models are synthesizable.
- Applications: design of RAM cells for the target technology, where only the generic parameters (width, depth, number of ports, (a)synchronous,...) need to be specified.
- Advantages
  - Explore Architectural Tradeoffs
  - Automatic generation of state machines
  - Fewer lines of VHDL code
  - Faster simulation than RTL
- Disadvantages
  - No timing information
  - Tools are not mature
  - May not be appropriate for non-algorithmic designs
RTL Synthesis

- RTL synthesis implements all functionality within a single clock cycle

- Behavioral synthesis automatically allocates the functionality across multiple clock cycles

Behavioral/RTL Synthesis

From Synopsys Behavioral Compiler Presentation
Register Transfer Level in VHDL

- Functional behavior is modeled with registered process (clocked process) and combinational process.
- RTL VHDL code contains some sort of structural information in addition to the functional behavior.

Gate Level in VHDL

- Contains a list of the gates components (e.g. ND2, NR2, AO6).
- Each single element of the circuit (e.g. U86) is instantiated as a component (e.g. ND2) and connected to corresponding signals (n192, n191, n188).
VHDL Language & Syntax (General)

-- Example VHDL Code --

```vhdl
signal mySignal: bit;
MYsignal <= '0',
    '1' after 10 ns,
    '0' after 20 ns,
    '1' after 30 ns;
```

- Signal assignment: `'<= '`
- User defined names:
  - Letters, numbers, underscores.
  - Start with a letter.
  - No VHDL keyword may be used.
  - Case insensitive
- List delimiter: `','`
- Statements are terminated by `';'` (may span multiple lines)
- Comments: `'-='` until end of line

VHDL Language & Syntax (Identifier)

- Normal Identifier:
  - Letters, numbers, underscores
  - Case insensitive.
  - The first character must be a letter.
  - The last character cannot be an underscore.
  - No two consecutive underscores.
  - VHDL reserved words may not be used as identifiers.

- Extended Identifier (VHDL93)
  - Enclosed in back slashes
  - Case sensitive
  - Graphical characters allowed
  - May contain spaced and consecutive underscores.
  - VHDL keywords allowed

MySignal_23 -- normal identifier
rdy, RDY, Rdy -- identical identifiers
vector & _vector -- X : special character
last of Zout -- X : white spaces
idle__state -- X : consecutive underscores
24th_signal -- X : begins with a numeral
open, register -- X : VHDL keywords

\mySignal_23\ -- extended identifier
\rdy\, \RDY\, \Rdy\ -- different identifiers
\vector\_\_vector\ -- legal
\last\of\ Zout\ -- legal
\idle\_\state\ -- legal
\24th\_signal\ -- legal
\open\, \register\ -- legal
Legal and Illegal Identifiers

- **Legal Identifiers:**
  - Uconn_huskies
  - ECE_252
  - Sel6B

- **Illegal Identifiers:**
  - _time_is_9am -- an identifier must start with a letter.
  - 8thsemester -- an identifier must start with a letter.
  - Homework#1 -- letter, digits, and underscore only.
  - final__example -- two underscore in succession not allowed
  - Entity -- keyword cannot be used as identifier
  - Time_out_ -- last character cannot be an underscore.

VHDL Reserved Words

<table>
<thead>
<tr>
<th>abs</th>
<th>disconnect</th>
<th>label</th>
<th>package</th>
<th>s1a</th>
</tr>
</thead>
<tbody>
<tr>
<td>access</td>
<td>downto</td>
<td>library</td>
<td>port</td>
<td>s1l</td>
</tr>
<tr>
<td>after</td>
<td>else</td>
<td>linkage</td>
<td>postponed</td>
<td>sra</td>
</tr>
<tr>
<td>alias</td>
<td>elsif</td>
<td>literal</td>
<td>procedure</td>
<td>srl</td>
</tr>
<tr>
<td>all</td>
<td>end</td>
<td>loop</td>
<td>process</td>
<td>subtype</td>
</tr>
<tr>
<td>and</td>
<td>entity</td>
<td>map</td>
<td>protected</td>
<td>then</td>
</tr>
<tr>
<td>architecture</td>
<td>exit</td>
<td>mod</td>
<td>pure</td>
<td>to</td>
</tr>
<tr>
<td>array</td>
<td>file</td>
<td>nand</td>
<td>range</td>
<td>transport</td>
</tr>
<tr>
<td>assert</td>
<td>for</td>
<td>new</td>
<td>record</td>
<td>type</td>
</tr>
<tr>
<td>attribute</td>
<td>function</td>
<td>next</td>
<td>register</td>
<td>unaffected</td>
</tr>
<tr>
<td>begin</td>
<td>generate</td>
<td>nor</td>
<td>reject</td>
<td>units</td>
</tr>
<tr>
<td>block</td>
<td>generic</td>
<td>not</td>
<td>rem</td>
<td>until</td>
</tr>
<tr>
<td>body</td>
<td>group</td>
<td>null</td>
<td>report</td>
<td>use</td>
</tr>
<tr>
<td>buffer</td>
<td>guarded</td>
<td>of</td>
<td>return</td>
<td>variable</td>
</tr>
<tr>
<td>bus</td>
<td>if</td>
<td>on</td>
<td>rol</td>
<td>wait</td>
</tr>
<tr>
<td>case</td>
<td>impure</td>
<td>open</td>
<td>ror</td>
<td>when</td>
</tr>
<tr>
<td>component</td>
<td>in</td>
<td>or</td>
<td>select</td>
<td>while</td>
</tr>
<tr>
<td>configuration</td>
<td>inertial</td>
<td>others</td>
<td>severity</td>
<td>with</td>
</tr>
<tr>
<td>constant</td>
<td>inout</td>
<td>out</td>
<td>shared</td>
<td>xnor</td>
</tr>
<tr>
<td>is</td>
<td>signal</td>
<td>xor</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
VHDL Structural Elements

- **Entity**: description of interface consisting of the port list.
  - The primary hardware abstraction in VHDL, analogous to a symbol in a block diagram.
- **Architecture**: description of the function of the corresponding module.
- **Process**: allows for a sequential execution of the assignments
- **Configuration**: used for simulation purposes.
- **Package**: hold the definition of commonly used data types, constants and subprograms.
- **Library**: the logical name of a collection of compiled VHDL units (object code).
  - Mapped by the simulation or synthesis tools.

Basic VHDL Concepts

- Interfaces
- Behavior
- Structure
- Test Benches
- Analysis, elaboration, simulation
- Synthesis
A VHDL Design Unit consists of an Entity Declaration and an Architecture Body.

**Entity Declaration:**
Names entity and defines interfaces between entity and its environment.

```
ENTITY entity_name IS
PORT ( name_list : mode type);
END entity_name ;
```

**Architecture Body:**
Establishes relationship between inputs and outputs of design.

```
ARCHITECTURE body_name OF entity_name IS
-- declarative_statements
BEGIN
-- activity_statements
END body_name;
```

### Entity statement

```
entity HALFADDER is
  port(  
    A, B: in bit;  
    SUM, CARRY: out bit);  
end HALFADDER;

entity ADDER is
  port(  
    A, B: in integer range 0 to 3;  
    SUM: out integer range 0 to 3;  
    CARRY: out bit );  
end ADDER;
```

- **Interface description**
- Port clause identifies ports used by "entity" to communicate with its environment.
- Port signals
  - Data types: bit, int, ...
  - Signal width: 0 to 3
  - Signal direction: in, out, ...
Architecture Body Statement

An architecture defines an entity's behavior from a simulation point of view.

Implementation of design

Always connected with a specified entity

• One entity can have several architectures
• Entity ports are available as signals within the architecture.
• Contains concurrent statements.

entity HALFADDER is
  port(
    A, B: in bit;
    SUM, CARRY: out bit);
end HALFADDER;

-- Architecture body
-- "Hadd" is user defined name
architecture Hadd of HALFADDER is begin
  SUM <= A xor B;
  CARRY <= A and B;
end architecture Hadd;

Architecture Body Structure

-- Declarative part:
• data types
• constants
• additional signals ("actual" signals)
• components
• ...

-- Statement part (after 'begin'):
• signal assignments
• processes
• component instantiations
• all concurrent statements that can be placed within the statement part..

-- architecture Body
architecture EXAMPLE of STRUCTURE is
  -- Declarative part
  subtype DIGIT is integer range 0 to 9;
  constant BASE: integer := 10;
  signal DIGIT_A, DIGIT_B: DIGIT;
  signal CARRY: DIGIT;
begin
  -- Statement part
  DIGIT_A <= 3;
  SUM <= DIGIT_A + DIGIT_B;
  DIGIT_B <= 7;
  CARRY <= 0 when SUM < BASE else 1;
end EXAMPLE ;
Example

Write a VHDL code for the circuit shown below. The inputs to the circuit are x1, x2, x3, and the output is f.

Example: VHDL code

```vhdl
ENTITY example1 IS
  PORT (x1, x2, x3: IN BIT;
        f: OUT BIT);
END example1;

ARCHITECTURE logicFunc OF example1 IS
BEGIN
  -- Architecture statement region
  f <= (x1 AND x2) NOR (NOT x2 AND x3);
END logicFunc;
```

- **logicFunc**: user-defined name