EECE-4740/5740 Advanced VHDL and FPGA Design

Lecture 3
Introduction to VHDL

Cristinel Ababei
Marquette University
Department of Electrical and Computer Engineering

Outline

- VHDL Overview
- VHDL Characteristics and Concepts
- Basic VHDL modelling
  - Entity declaration
  - Architecture declaration
- Behavioural vs. Structural description in VHDL
Typical design flow

VHDL overview

- What does VHDL stand for?
  - Very High Speed Integrated Circuit (VHSIC) Hardware Description Language
- VHDL is a formal language for specifying the behavior and structure of a digital circuit
  - Concurrent and sequential statements
  - Machine-readable specification
  - Man- and machine-readable documentation
- Initially developed under DOD auspices, later standardized as IEEE standards 1076-1987, 1076-1993, & 1076-1164 (standard logic data type)
- A concurrent language, initially aimed at simulation, later at synthesis
- Syntax similar to ADA and Pascal
- Verilog is another, equally popular, hardware description language (HDL)
Hardware Description Languages

- Both VHDL and Verilog are hardware description languages.
- They describe hardware!
- They are not software programming languages.

Application of HDL

- HDL offers design reuse capability
  - The corresponding HDL model can be reused in several designs/projects.
  - Frequently needed function blocks (macros) are collected in model libraries.
Range of use

- **Behavioural level:**
  - Functional description of the design
  - Easy to describe in VHDL
  - Useful especially for simulation purposes
  - May not necessarily be synthesizable

Abstraction levels in Digital Design

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

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

- **Logic level:**
  - Design is represented as a netlist of interconnected logic gates (AND, OR, NOT,...) and storage elements

- **Layout level (not really relevant to VHDL discussion):**
  - Logic cells of target technology are placed on the chip and connections are routed
  - After layout is verified, the design is ready for the manufacturing/fabrication

Information Content of Abstraction Levels

- **Behaviour:** Functional timing behaviour “after 10 ns, signal A switches to ‘1’”
- **RTL:** NO time, Clock, function, events
- **Logic:** Gate delays
- **Layout:** Path delays
VHDL design unit – a quick intro

- A VHDL Design Unit consists of:
  1) Entity declaration
  2) Architecture

### 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;
```

1) Entity Declaration

- Names entity and defines interfaces between entity and its environment.

```
entity entity-name is port ( 
  port-name-A: mode type;
  port-name-B: mode type;
  port-name-C: mode type;
  ...
);
end [entity] [entity-name];
```
Each I/O signal in the entity statement is referred to as a **port**.

- A port is analogous to a pin on a schematic.
- A port is a data object.
- Can be assigned values.
- Can be used in expressions.

**Port**

- **Mode** describes the direction in which data is transferred through a port.
- There are 4 different modes:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>Data only flows into the entity (input)</td>
</tr>
<tr>
<td>out</td>
<td>Data only flows out of the entity (output)</td>
</tr>
<tr>
<td>inout</td>
<td>Data flows into or out of the entity (bidirectional)</td>
</tr>
<tr>
<td>buffer</td>
<td>Used for internal feedback</td>
</tr>
</tbody>
</table>
VHDL is a strongly typed language

Data objects of different types cannot be assigned to one another without the use of a type-conversion function.

There are two broad categories of data types:

- Scalar - stores a single value
- Composite - stores multiple values

VHDL data types include:

<table>
<thead>
<tr>
<th>scalar</th>
<th>bit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>boolean</td>
</tr>
<tr>
<td></td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>character</td>
</tr>
<tr>
<td></td>
<td>std_ulogic</td>
</tr>
<tr>
<td></td>
<td>std_logic</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>composite</th>
<th>bit_vector</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>std_ulogic_vector</td>
</tr>
<tr>
<td></td>
<td>std_logic_vector</td>
</tr>
</tbody>
</table>

The most useful types for synthesis and simulation, provided by the IEEE std_logic_1164 package:

- std_logic
- std_ulogic
- std_logic_vector
- std_ulogic_vector

See Appendix A for difference between std_logic and std_ulogic

IEEE Standard Logic Types

- Use of two-valued logic (bit and bit_vector) is generally not sufficient to simulate digital systems.
- In addition to 0 and 1, Z (high-impedance), X (unknown), and U (uninitialized) are often used in digital system simulation.
- The IEEE standard 1164 defines the std_logic type that has nine values:
  - 0, 1, Z, X, U, W, L, H, -
Entity Declaration - example

entity FULL_ADDER is
port (
    A, B, Cin: in std_logic;
    S:        out std_logic;
    Cout:     out std_logic;
end FULL_ADDER;

2) Architecture Declaration

- Establishes relationship between inputs and outputs of design.

architecture architecture-name of entity-name is
[declarations]
begin
    architecture body
end [architecture][architecture-name];
Architecture body

- Several different models or styles may be used in the architecture body including:
  - Behavioral
    - Dataflow
    - Algorithmic
  - Structural
- These models allow to describe the design at different levels of abstraction.

Architecture statement

- One or more architecture statements may be associated with an entity statement.
  - Only one may be referenced at a time.
- Declarations
  - Signals and components.
- Architecture body
  - Statements that describe the functionality of the design (i.e., the circuit).
Architecture Declaration – example

```vhdl
architecture RTL of FULL_ADDER is
begin
  S <= A xor B xor Cin;
  Cout <= (A and B) or (A and Cin) or (B and Cin);
end RTL;
```

Models/styles of description in VHDL

- Behavioral
  - Dataflow
  - Algorithmic
- Structural
- RTL
1) Behavioral description in VHDL

**Specification:**

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i_1 )</td>
<td>( o )</td>
</tr>
<tr>
<td>( i_2 )</td>
<td></td>
</tr>
<tr>
<td>( i_3 )</td>
<td></td>
</tr>
</tbody>
</table>

\( o \leq transport \; i_1 + i_2 \times i_3 \) after 100 ns;

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

**Behavioral description in VHDL**

- Specify a set of statements to model the function, or behavior, of the design.
- **Dataflow:** uses concurrent statements
  - Concurrent statements:
    - Are executed at the same time; they mimic the actual hardware parallelism (processes, signal assignment)
    - Order is unimportant
- **Algorithmic:** uses sequential statements
  - Sequential statements:
    - Are executed in sequence (if, case, loops – while, for – assertion)
    - Order is very important
Behavioral synthesis

- Advantages
  - Easy to write HDL code; fewer lines of VHDL code
  - Useful especially for automatic generation of state machines
  - Faster simulation than RTL
- Disadvantages
  - May not be synthesizable

2) Structural description in VHDL

- Specify a set of statements to instantiate and interconnect the components necessary for the design.
- Components are defined separately.
- Signals are used to interconnect components.
- Advantages
  - Helps to describe a design hierarchically
  - Offers better control of circuit timing
  - Allows user to focus design optimization efforts on specific parts of design
- Disadvantages
  - Requires knowledge of internal structure of design
  - More VHDL code to write
Gate level in VHDL – an example/form of structural description

- 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).

3) RTL description in VHDL

- Most realistic circuits combine a control-path or controller and a datapath to perform some computation.
- In this case the description style in VHDL is closely related to the so called RTL design methodology, in which operations are specified as data manipulation and transfer among a collection of registers.
- For example, the use of the FSMD model is especially recommended whenever the structure of the datapath is important.
- This description style in VHDL can be regarded as a combination of behavioral and structural descriptions.
FSM as an example of the simplest RTL description in VHDL

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

Example: simple combinational logic circuit
Example: entity

```vhdl
entity comb_logic_ckt_1 is
  Port ( A, B, C : in STD_LOGIC;
         F : out STD_LOGIC);
end comb_logic_ckt_1;
```

Example: architecture #1

```vhdl
architecture Boolean_Exp of comb_logic_ckt_1 is begin
  F <= ( not(A) and B and C ) or
       ( A and not(B) and C ) or
       ( A and B and not(C) );
end Boolean_Exp;
```
Example: architecture #2

architecture Truth_table of comb_logic_ckt_1 is
begin
    F <= '0' when ( A = '0' ) and ( B = '0' ) and ( C = '0' ) else
       '0' when ( A = '0' ) and ( B = '0' ) and ( C = '1' ) else
       '0' when ( A = '0' ) and ( B = '1' ) and ( C = '0' ) else
       '1' when ( A = '0' ) and ( B = '1' ) and ( C = '1' ) else
       '0' when ( A = '1' ) and ( B = '0' ) and ( C = '0' ) else
       '1' when ( A = '1' ) and ( B = '0' ) and ( C = '1' ) else
       '1' when ( A = '1' ) and ( B = '1' ) and ( C = '0' ) else
       '0' when ( A = '1' ) and ( B = '1' ) and ( C = '1' ) else
       '0';
end Truth_table;

Example: architecture #3

architecture Logic_gates of comb_logic_ckt_1 is
-- Component Declarations
-- components are defined in a VHDL package
component AND3
    port( inA, inB, inC: in std_logic;
         outF: out std_logic );
end component;

component OR3
    port( inA, inB, inC: in std_logic;
         outF: out std_logic );
end component;

component NOT1
    port( inA: in std_logic;
         outF: out std_logic );
end component;

-- Signal Declarations
-- used to interconnect the gates in the circuit (i.e. "wires")
signal out1, out2, out3: std_logic;
signal out4, out5, out6: std_logic;
Example: architecture #3 (continued)

begin
  NOTgate1: NOT1 port map( inA => A, outF => out1 );
  NOTgate2: NOT1 port map( B, out2 );
  NOTgate3: NOT1 port map( inA => C, outF => out3 );
  ANDgate1: AND3 port map( inA => out1, inB => B, inC => C, outF => out4 );
  ANDgate2: AND3 port map( A, out2, C, out5 );
  ANDgate3: AND3 port map( inA => A, inB => B, inC => out3, outF => out6 );
  ORgate1: OR3 port map( inA => out4, inB => out5, inC => out6, outF => F );
end Logic_gates;

Structural Model

VHDL Language & Syntax (General)

- 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: " -- " till 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:**
  - Enclosed in backslashes
  - Case sensitive
  - Graphical characters allowed
  - May contain spaced and consecutive underscores.
  - VHDL keywords allowed.

- **Legal Identifiers:**
  - my_beautiful_signal
  - EE_459_500
  - 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>sla</th>
</tr>
</thead>
<tbody>
<tr>
<td>access</td>
<td>downto</td>
<td>library</td>
<td>port</td>
<td>sli</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></td>
<td>is</td>
<td></td>
<td>signal</td>
<td>xor</td>
</tr>
</tbody>
</table>

**VHDL information**

- **Recommended books on VHDL or the use of VHDL:**
  - Peter J. Ashenden, The Student’s Guide to VHDL, Morgan Kaufmann.
  - Peter J. Ashenden, The Designer's Guide to VHDL, Morgan Kaufmann.
  - S. Yalamanchili, Introductory VHDL from Simulation to Synthesis, Prentice Hall.

- **Useful websites** – see the links provided at:
Summary

- VHDL is a hardware description language
- It has syntax (structure) and semantics (meaning)
- Entity and architecture(s) declaration
- Models/styles of description in HDL: Behavioral, Structural, RTL

Appendix A: What’s the difference between std_logic and std_ulogic?

- The library ieee.std_logic_1164 has two types
  std_logic and std_ulogic
- To understand the difference, consider this circuit

where the blue signal line has more than one driver attached to it? (e.g., it’s a bus)
- How do we set up our model so that the simulator knows the ‘rules’?
  - which signal overrides the others
    - or
  - how the signals combine together
Contending drivers

- Remember that VHDL knows nothing about the IEEE 1164 rules
  - To VHDL, the only primitive operations are those of a ‘normal’ programming language
    - addition, subtraction, etc
    - assignment
      - It does distinguish between signal and variable assignment, but only with respect to the timing of assignment of new values!

- ieee.std_logic_1164 is **NOT** part of the VHDL standard
  - So when two std_logic values are applied to the same signal (i.e., wire), a VHDL simulator has to know that
    - ‘1’ and ‘0’ lead to ‘X’
      - etc.

Unresolved signals

- **std_ulogic** is an *unresolved* type
  - It is an error to define a model in which two drivers can set the value of an unresolved signal
    - because
      - there is no *resolution function* associated with the signal that can be invoked to determine which driver overrides the other
  - It is defined simply:
    ```
    TYPE std_ulogic IS ('U','X','0','1','Z','W','L','H', '-');
    ```
  - *i.e.,* it is an enumerated type with possible values: ‘U’, ...
  - This says nothing about the behavior of std_ulogic signals
    - Their behavior is encoded in the functions (and, or, ...) that take std_ulogic arguments

*On the other hand,*

- **std_logic** is an *resolved* type
Unresolved signals

On the other hand,

- **std_logic** is an **resolved** type
  - It is defined:
    ```
    SUBTYPE std_logic IS resolved std_ulogic;
    
    Note that there is a function definition just preceding this type:
    
    FUNCTION resolved( s: std_ulogic_vector )
    RETURN std_ulogic;
    SUBTYPE std_logic IS resolved std_ulogic;
    ```
  - Thus **resolved** is a function that takes a vector of **std_ulogic** elements and returns a value of **std_ulogic** type
  - This function is called a **resolution function**
    - It is called whenever two or more sources (signal assignments) drive a **std_logic** signal

Resolution functions

- Any **resolved signal** (*i.e.*, one that may be driven by two sources) is defined by a type that has a resolution function associated with it
  - A resolved type is a subtype
    - It can resolve a conflict of multiple instances of the parent type
  - The name of the resolution function immediately precedes the name of the type being resolved
  - The resolution function’s
    - argument is a vector of elements of the type being resolved
      - The simulator will place the actual values to be resolved in this vector and call the resolution function
        - e.g., with 3 drivers for a **std_logic** signal, the argument to resolved might be (‘2’, ‘H’, ‘1’) which should return ‘1’
    - return value is the parent type
      - It will determine which of the values of the parent type result when the vector of signal values is applied
The simplest way to implement a resolution function uses a table

\[ e.g., \text{for } \texttt{std_logic} \]

```cpp
TYPE std_logic_table IS ARRAY(std_ulogic,std_ulogic) OF std_ulogic;
CONSTANT resolution_table : std_logic_table := (
  -- U   X   0   1   Z   W   L   H   --
  ( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), -- U
  ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- X
  ( 'U', 'X', '0', 'X', '0', '0', '0', '0', '0' ), -- 0
  ( 'U', 'X', 'X', '1', '1', '1', '1', '1', '1' ), -- 1
  ( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X' ), -- Z
  ... );
```

The function `resolved` is now very simple

```cpp
FUNCTION resolved ( s: std_ulogic_vector ) RETURN std_ulogic IS
  VARIABLE result : std_logic := 'Z'; -- default, weakest
BEGIN
  IF ( s'LENGTH = 1 ) THEN RETURN s(s'LOW);
ELSE
  FOR k IN s'RANGE LOOP
    -- Take each signal in turn and determine the result of
    -- combining it with the previous result
    result := resolution_table(result, s(k));
  END LOOP;
END IF;
RETURN result;
END resolved;
```
### Writing resolution functions

- **You may never need to!**
- `std_logic_1164` defines the most commonly needed one!

But,

1. You may be using `integer` types instead of `std_logic_vector` in a model of a processor for
   - convenience
   - speed in simulation
   - ...

   You will need to define a resolved integer types if your model has a bus with multiple drivers in it
   - You will need to have a convention for 'disconnecting' a driver, e.g., setting a driver to emit 0 when it's not driving the bus (where you would drive a 'Z' with `std_logic`)
   - You can also explicitly disconnect a driver with VHDL’s `DISCONNECT` statement

### Resolution functions

2. You may have defined an abstract type
   - You (correctly) don’t want to be bothered with implementation details yet
   - Your bus is a collection of signals (address, data, command, etc); you have a type for each one; so the bus itself is defined as a VHDL RECORD
   - The synthesizer will eventually convert it to logic for you!
   - ...
   - Again you will need a resolution function

3. ...
Simulation speed

- **std_ulogic** does not have a resolution function associated with it
  - It should use less simulation time (i.e., run faster) than **std_logic**
  - With **std_logic**, the simulator may end up checking for (or calling) the resolution function for every assignment

- For simulation purposes **std_ulogic** is recommended wherever possible
  - It may save simulation time
  - **std_logic** is a subtype, so it is possible to convert between them whenever necessary
    - ```res_signal <= std_logic(unres_signal);```  
  - However, lots of people stick to **std_logic** for different reasons (laziness?, compatibility with IPs, etc.)