Basic Language Constructs of VHDL

Outline

1. Basic VHDL program
2. Lexical elements and program format
3. Objects
4. Data type and operators

Design unit

1. Basic VHDL program

- Building blocks in a VHDL program
- Each design unit is analyzed and stored independently
- Types of design unit:
  - entity declaration
  - architecture body
  - package declaration
  - package body
  - configuration

Entity declaration

- Simplified syntax

```vhdl
entity entity_name is
  port( 
    port_names: mode data_type;
    port_names: mode data_type;
    ...
    port_names: mode data_type 
  );
end entity_name;
```

- mode:
  - in: flow into the circuit
  - out: flow out of the circuit
  - inout: bi-directional

- E.g.

```vhdl
entity even_detector is
  port( 
    a: in std_logic_vector(2 downto 0);
    even: out std_logic);
end even_detector;
```
• A common mistake with mode

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity node_demo is
port(a, b: in std_logic;
x, y: out std_logic);
end node_demo;
architecture wrong_arch of node_demo is
begin
  x <= a and b;
y <= not x;
end wrong_arch;
```

• Fix: use an internal signal

```vhdl
architecture ok_arch of node_demo is
signal ab: std_logic;
begin
  ab <= a and b;
x <= ab;
y <= not ab;
end ok_arch;
```

### Architecture body

• Simplified syntax

```vhdl
architecture arch_name of entity_name is
declarations;
begin
  concurrent statement;
  concurrent statement;
  
end arch_name;
```

• An entity declaration can be associated with multiple architecture bodies

### Other design units

• Package declaration/body:
  – collection of commonly used items, such as data types, subprograms and components

• Configuration:
  – specify which architecture body is to be bound with the entity declaration

### VHDL Library

• A place to store the analyzed design units

• Normally mapped to a directory in host computer

• Software define the mapping between the symbolic library and physical location

• Default library: "work"

• Library "ieee" is used for many ieee packages
• E.g.

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

• Line 1: invoke a library named ieee
• Line 2: makes std_logic_1164 package visible to the subsequent design units
• The package is normally needed for the std_logic/std_logic_vector data type

### Processing of VHDL code

#### Analysis
- Performed on “design unit” basis
- Check the syntax and translate the unit into an intermediate form
- Store it in a library

#### Elaboration
- Bind architecture body with entity
- Substitute the instantiated components with architecture description
- Create a “flattened” description

#### Execution
- Simulation or synthesis

### Lexical elements

#### Lexical element:
- Basic syntactical units in a VHDL program

#### Types of Lexical elements:
- Comments
- Identifiers
- Reserved words
- Numbers
- Characters
- Strings

#### Comments

- Starts with `--`
- Just for clarity
- e.g.,

```vhdl
-- example to show the caveat of the not node
architecture arch of node_dem is
    signal ab: std_logic; -- ab is the internal signal
begin
    ab <= a and b;
    x <= ab;
    -- ab connected to the x output
    y <= not ab;
end arch;
```

#### Identifier

- Identifier is the name of an object
- Basic rules:
  - Can only contain alphabetic letters, decimal digits and underscore
  - The first character must be a letter
  - The last character cannot be an underscore
  - Two successive underscores are not allowed
• Valid examples:
  A10, next_state, NextState, mem_addr_enable
• Invalid examples:
  sig#3, _X10, 7segment, X10_, hi__there
• VHDL is case insensitive:
  – Following identifiers are the same:
    nextstate, NextState, NEXTSTATE, nEXTSTATE

Reserved words

Library

use ieee.std_logic_1164.all;
entity even_detector is
  port(
    a : in std_logic_vector(2 downto 0);
    even : out std_logic);
end even_detector;

architecture eg_arch of even_detector is
  signal p1, p2, p3, p4 : std_logic;
begnin
  even <= (p1 or p2) or (p3 or p4);
  p1 <= (not a(0)) and (not a(1)) and (not a(2));
  p2 <= (not a(0)) and a(1) and a(2);
  p3 <= a(0) and (not a(1)) and a(2);
  p4 <= a(0) and a(1) and (not a(2));
end eg_arch;

Program format

• VHDL is “free-format”: blank space, tab, new-line can be freely inserted
  • e.g., the following are the same

---

A good "header"

---
3. Objects

Objects

- A named item that hold a value of specific data type
- Four kinds of objects
  - Signal
  - Variable
  - Constant
  - File (cannot be synthesized)
- Related construct
  - Alias

Objects

- Signal
  - Declared in the architecture body's declaration section
  - Signal declaration:
    ```
    signal signal_name, signal_name, ... : data_type
    ```
  - Signal assignment:
    ```
    signal_name <= projected_waveform;
    ```
  - Ports in entity declaration are considered as signals
  - Can be interpreted as wires or "wires with memory" (i.e., FFs, latches etc.)

Objects

- Variable
  - Declared and used inside a process
  - Variable declaration:
    ```
    variable variable_name, ... : data_type
    ```
  - Variable assignment:
    ```
    variable_name := value_expression;
    ```
  - Contains no "timing info" (immediate assignment)
  - Used as in traditional PL: a "symbolic memory location" where a value can be stored and modified
  - No direct hardware counterpart

Objects

- Constant
  - Value cannot be changed
  - Constant declaration:
    ```
    constant const_name, ... : data_type := value_expression
    ```
  - Used to enhance readability
    - E.g.,
    ```
    constant BUS_WIDTH: integer := 32;
    constant BUS_BYTES: integer := BUS_WIDTH / 8;
    ```
  - It is a good idea to avoid "hard literals"

Objects

- Architecture
  ```
  architecture behl_arch of even_detector is
    signal odd: std_logic;
  begin
    ..
    tnp := '0';
    for i in 2 downto 0 loop
      tnp := tnp xor a(i);
    end loop;
    ..
  end architecture;
  ```
Alias

- Not an object
- Alternative name for an object
- Used to enhance readability
  - E.g.,

```vhdl
signal: word: std_logic_vector(16 downto 0);
alias ep: std_logic_vector(6 downto 0) is word(15 downto 0);
alias reg: std_logic_vector(2 downto 0) is word(8 downto 0);
alias reg2: std_logic_vector(2 downto 0) is word(5 downto 3);
alias reg3: std_logic_vector(2 downto 0) is word(2 downto 0);
```

4. Data type and operators

- Standard VHDL
- IEEE1164_std_logic package
- IEEE numeric_std package

Data type

- Definition of data type
  - A set of values that an object can assume.
  - A set of operations that can be performed on objects of this data type.
- VHDL is a strongly-typed language
  - An object can only be assigned with a value of its type
  - Only the operations defined with the data type can be performed on the object

Data types in standard VHDL

- integer:
  - Minimal range: -(2^31-1) to 2^31-1
  - Two subtypes: natural, positive
- boolean: (false, true)
- bit: ('0', '1')
  - Not capable enough
- bit_vector: a one-dimensional array of bit

Operators in standard VHDL

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Data type of operand a</th>
<th>Data type of operand b</th>
<th>Data type of result</th>
</tr>
</thead>
<tbody>
<tr>
<td>a + b</td>
<td>Addition</td>
<td>integer</td>
<td>integer</td>
<td>integer</td>
</tr>
<tr>
<td>a - b</td>
<td>Subtraction</td>
<td>integer</td>
<td>integer</td>
<td>integer</td>
</tr>
<tr>
<td>a * b</td>
<td>Multiplication</td>
<td>integer</td>
<td>integer</td>
<td>integer</td>
</tr>
<tr>
<td>a / b</td>
<td>Division</td>
<td>integer</td>
<td>integer</td>
<td>integer</td>
</tr>
<tr>
<td>a rem b</td>
<td>Remainder</td>
<td>integer</td>
<td>integer</td>
<td>integer</td>
</tr>
<tr>
<td>a &lt;&lt; b</td>
<td>Shift left logical</td>
<td>integer</td>
<td>integer</td>
<td>bit_vector</td>
</tr>
<tr>
<td>a shl b</td>
<td>Shift left arithmetic</td>
<td>integer</td>
<td>integer</td>
<td>bit_vector</td>
</tr>
<tr>
<td>a &gt;&gt; b</td>
<td>Shift right</td>
<td>integer</td>
<td>integer</td>
<td>bit_vector</td>
</tr>
<tr>
<td>a shr b</td>
<td>Shift right arithmetic</td>
<td>integer</td>
<td>integer</td>
<td>bit_vector</td>
</tr>
<tr>
<td>a and b</td>
<td>Logical AND</td>
<td>boolean, bit</td>
<td>boolean, bit</td>
<td>boolean, bit_vector</td>
</tr>
<tr>
<td>a or b</td>
<td>Logical OR</td>
<td>boolean, bit</td>
<td>boolean, bit</td>
<td>boolean, bit_vector</td>
</tr>
<tr>
<td>a xor b</td>
<td>Exclusive OR</td>
<td>boolean, bit</td>
<td>boolean, bit</td>
<td>boolean, bit_vector</td>
</tr>
</tbody>
</table>

RTL Hardware Design
by P. Chu

Chapter 3
IEEE std_logic_1164 package

- What’s wrong with bit?
- New data type: std_logic, std_logic_vector

std_logic:
- 9 values: ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-')
  - '0', '1': forcing logic 0 and forcing logic 1
  - 'Z': high-impedance, as in a tri-state buffer.
  - 'L', 'H': weak logic 0 and weak logic 1, as in wired-logic
  - 'X', 'W': "unknown" and "weak unknown"
  - 'U': for uninitialized
  - '-': don't-care.

std_logic_vector
- an array of elements with std_logic data type
- Imply a bus

E.g.,
signal a: std_logic_vector(7 downto 0);
- Another form (less desired)
signal a: std_logic_vector(0 to 7);

Need to invoke package to use the data type:
library ieee;
use ieee.std_logic_1164.all;

Overloaded operator
IEEE std_logic_1164 package

- Which standard VHDL operators can be applied to std_logic and std_logic_vector?
- Overloading: same operator of different data types
- Overloaded operators in std_logic_1164 package

<table>
<thead>
<tr>
<th>overloaded operator</th>
<th>data type of operand a</th>
<th>data type of operand b</th>
<th>data type of result</th>
</tr>
</thead>
<tbody>
<tr>
<td>a and b</td>
<td>std_logic_vector</td>
<td>std_logic</td>
<td>same as a</td>
</tr>
<tr>
<td>a or b</td>
<td>std_logic_vector</td>
<td>std_logic</td>
<td>same as a</td>
</tr>
<tr>
<td>a and b</td>
<td>std_logic_vector</td>
<td>std_logic_vector</td>
<td>same as a</td>
</tr>
<tr>
<td>a xor b</td>
<td>std_logic</td>
<td>std_logic</td>
<td>same as a</td>
</tr>
<tr>
<td>a or b</td>
<td>std_logic</td>
<td>std_logic</td>
<td>same as a</td>
</tr>
</tbody>
</table>

Type conversion function in std_logic_1164 package:

<table>
<thead>
<tr>
<th>function</th>
<th>data type of operand a</th>
<th>data type of result</th>
</tr>
</thead>
<tbody>
<tr>
<td>to_bit(a)</td>
<td>std_logic</td>
<td>bit</td>
</tr>
<tr>
<td>to_std_logic(a)</td>
<td>bit</td>
<td>std_logic integer</td>
</tr>
<tr>
<td>to_bitvector(a)</td>
<td>std_logic_vector</td>
<td>bitvector</td>
</tr>
<tr>
<td>to_std_logic_vector(a)</td>
<td>bitvector</td>
<td>std_logic_vector</td>
</tr>
</tbody>
</table>

Operators over an array data type

- Relational operators for array
  - operands must have the same element type but their lengths may differ
  - Two arrays are compared element by element, form the left most element
  - All following returns true
    - "011"="011", "011">"010", "011">"00010", "0110">"011"
Array aggregate

- Aggregate is a VHDL construct to assign a value to an array-typed object
- E.g.,
  a <= "10100000";
  a <= (7=>'1', 6=>'0', 0=>'0', 1=>'0', 5=>'1', 4=>'0', 3=>'0', 2=>'1');
  a <= (7=>'1', others=>'0');
- E.g.,
  a <= "00000000";
  a <= (others=>'0');

IEEE numeric_std package

- How to infer arithmetic operators?
- In standard VHDL:
  
  ```vhdl
  signal a, b, sum: integer;
  ...
  sum <= a + b;
  ```

- What’s wrong with integer data type?

Overloaded operators in IEEE numeric_std package

- IEEE numeric_std package: define integer as an array of elements of std_logic
- Two new data types: unsigned, signed
- The array interpreted as an unsigned or signed binary number
- E.g.,
  ```vhdl
  signal x, y: signed(15 downto 0);
  ```

- Need invoke package to use the data type
  ```vhdl
  library ieee;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;
  ```

- E.g.,
  ```vhdl
  signal a, b, c, d: unsigned(7 downto 0);
  ...
  a <= b + c;
  d <= b + 1;
  e <= (5 + a + b) - c;
  ```
New functions in IEEE numeric_std package

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Data type of operand a</th>
<th>Data type of operand b</th>
<th>Data type of result</th>
</tr>
</thead>
<tbody>
<tr>
<td>shift_left(a,b)</td>
<td>shift left</td>
<td>std_logic_vector</td>
<td>std_logic_vector</td>
<td>natural</td>
</tr>
<tr>
<td>shift_right(a,b)</td>
<td>shift_right</td>
<td>std_logic_vector</td>
<td>std_logic_vector</td>
<td>natural</td>
</tr>
<tr>
<td>rotate_left(a,b)</td>
<td>rotate_left</td>
<td>std_logic_vector</td>
<td>std_logic_vector</td>
<td>natural</td>
</tr>
<tr>
<td>rotate_right(a,b)</td>
<td>rotate_right</td>
<td>std_logic_vector</td>
<td>std_logic_vector</td>
<td>natural</td>
</tr>
<tr>
<td>resize(a,b)</td>
<td>resize</td>
<td>std_logic_vector</td>
<td>std_logic_vector</td>
<td>natural</td>
</tr>
<tr>
<td>std_cast(a,b)</td>
<td>std_cast</td>
<td>std_logic_vector</td>
<td>std_logic_vector</td>
<td>std_logic_vector</td>
</tr>
<tr>
<td>to_integer(a)</td>
<td>to_integer</td>
<td>std_logic_vector</td>
<td>std_logic_vector</td>
<td>integer</td>
</tr>
<tr>
<td>to_unsigned(a)</td>
<td>to_unsigned</td>
<td>std_logic_vector</td>
<td>std_logic_vector</td>
<td>unsigned</td>
</tr>
<tr>
<td>to_signed(a)</td>
<td>to_signed</td>
<td>std_logic_vector</td>
<td>std_logic_vector</td>
<td>signed</td>
</tr>
</tbody>
</table>

Type conversion

- Std_logic_vector, unsigned, signed are defined as an array of element of std_logic
- They considered as three different data types in VHDL
  - Type conversion between data types:
    - type conversion function
    - Type casting (for "closely related" data types)

Type conversion between number-related data types

<table>
<thead>
<tr>
<th>Data type of a</th>
<th>To data type</th>
<th>Conversion function / Type casting</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned, signed</td>
<td>std_logic_vector</td>
<td>std_logic_vector</td>
</tr>
<tr>
<td>unsigned, std_logic_vector</td>
<td>unsigned</td>
<td>unsigned</td>
</tr>
<tr>
<td>unsigned, signed</td>
<td>std_logic_vector</td>
<td>std_logic_vector</td>
</tr>
<tr>
<td>natural</td>
<td>integer</td>
<td>to_integer(a)</td>
</tr>
<tr>
<td>integer</td>
<td>signed</td>
<td>to_signed(a, size)</td>
</tr>
<tr>
<td>signed</td>
<td>unsigned</td>
<td>to_unsigned(a, size)</td>
</tr>
</tbody>
</table>

E.g.

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

signal s1, s2, s3, s4, s5, s6: std_logic_vector(3 downto 0);
signal u1, u2, u3, u4, u6, u7: unsigned(3 downto 0);
signal sg: signed(3 downto 0);
```

- Ok
  ```vhdl
  u3 <= u2 + u1;  -- ok, both operands unsigned
  u4 <= u2 + 1;   -- ok, operands unsigned and natural
  ```
- Wrong
  ```vhdl
  u5 <= sg;   -- type mismatch
  u6 <= 5;    -- type mismatch
  ```
- Fix
  ```vhdl
  u5 <= unsigned(sg);   -- type casting
  u6 <= to_unsigned(5,4); -- conversion function
  ```
Non-IEEE package

- Packagea by Synopsys
- std_logic_arith:
  - Similar to numeric_std
  - New data types: unsigned, signed
  - Details are different
- std_logic_unsigned/ std_logic_signed
  - Treat std_logic_vector as unsigned and signed numbers
  - i.e., overload std_logic_vector with arith operations

- Software vendors frequently store them in ieee library:
  - E.g.,
    ```
    library ieee;
    use ieee.std_logic_1164.all;
    use ieee.std_arith_unsigned.all;
    ...
    signal s1, s2, s3, s4, s5, s6: std_logic_vector(3 downto 0);
    ...
    s5 <= s2 + s1; -- ok, + overloaded with std_logic_vector
    s6 <= s2 + 1;  -- ok, + overloaded with std_logic_vector
    ```

- Only one of the std_logic_unsigned and std_logic_signed packages can be used
- The std_logic_unsigned/std_logic_signed packages beat the motivation behind a strongly-typed language
- Numeric_std is preferred