Chapter 3
Basic Language Constructs of VHDL

Dr. Curt Nelson
Engr433 – Digital Design

Outline

• Basic VHDL programs;
• Lexical elements and program format;
• Objects;
• Data type and operators.
Design Unit

• Each design unit is analyzed and stored independently
• Types of design units
  – Entity declaration;
  – Architecture body;
  – Package declaration;
  – Package body;
  – Configuration.

Entity Declaration

• General 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;
```
Entity Declaration

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

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

A Common Mistake

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

- Use an internal signal

```vhdl
architecture ok_arch of mode_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;
  concurrent statement;
  ...
end arch_name;
```

- An entity declaration can be associated with multiple architecture bodies.
Example Architecture Body

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

Other Design Units

- Package declaration/body
  - collection of commonly used items, such as data types, subprograms and components.
- Configuration
  - specifies which architecture body is to be bound with the entity declaration.
VHDL Library

- A place to store the design units;
- Normally mapped to a directory in host computer;
- Software defines the mapping between the symbolic library and physical location;
- Default library is named `work` and is created in your project directory;
- Library “ieee” is used for many ieee packages.

VHDL Library Example

```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

• Start with - -
• Just for clarity

---- Example to show the caveat of the out mode ----

architecture arch of mode Demo 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 eg_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.
Identifier Examples

- 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

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

- **Number**
  - Integer: 0, 1234, 98E7
  - Real: 0.0, 1.23456 or 9.87E6
  - Base 2: 2#101101#

- **Character**
  - ‘A’, ‘Z’, ‘1’

- **Strings**
  - “Hello”, “101101”

- **Notes**
  - 0 and ‘0’ are different
  - 2#101101# and “101101” are different

Program Format

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

```vhdl
library ieee; 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; begin 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 - Example

library ieee;
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;
begin
    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;

A Good Program Header

*******************************************************************************
----------
Author: p chu
File: even_det.vhd
Design units:
    entity even_detector
    function: check even # of 1s from input
    input: a
    output: even
    architecture sop_arch:
    truth-table based sum-of-products
    implementation
Library/package:
    ieee.std_logic_1164: to use std_logic
Synthesis and verification:
    Synthesis software: ...
    Options/script: ...
    Target technology: ...
    Test bench: even_detector_tb
Revision history
Version 1.0:
Date: 9/2003
Comments: Original
*******************************************************************************
Objects

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

Signal

- Declared in the architecture body's declaration section
- Signal declaration
  \[
  \text{signal} \text{ signal\_name, signal\_name, ... : data\_type}
  \]
- Signal assignment
  \[
  \text{signal\_name} <= \text{projected\_waveform}
  \]
- Ports in entity declaration are considered as signals
  - Can be interpreted as wires or “wires with memory” (i.e., FFs, latches, etc.)
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 programming languages - a “symbolic memory location” where a value can be stored and modified.
- No direct hardware counterpart.

Constant

- Value cannot be changed.
- Constant declaration
  
  `constant` const_name, ... : data_type := value_expression

- Used to enhance readability.

```haskell
const BUS_WIDTH: integer := 32;
const BUS_BYTES: integer := BUS_WIDTH / 8;
```
Try to Avoid Hard Literals

```vhdl
architecture beh1_arch of even_detector is
  signal odd: std_logic;
begin
  tmp := '0';
  for i in 2 downto 0 loop
    tmp := tmp xor a(i);
  end loop;
end architecture beh1_arch;
```

```vhdl
architecture beh1_arch of even_detector is
  signal odd: std_logic;
  constant BUS_WIDTH: integer := 3;
begin
  tmp := '0';
  for i in (BUS_WIDTH-1) downto 0 loop
    tmp := tmp xor a(i);
  end loop;
end architecture beh1_arch;
```

Alias

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

```vhdl
signal: word: std_logic_vector(15 downto 0);
alias op: std_logic_vector(6 downto 0) is word(15 downto 9);
alias reg1: std_logic_vector(2 downto 0) is word(8 downto 6);
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);
```
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 (need more options as we will see later).
- Bit_vector - a one-dimensional array of type 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>exponentiation</td>
<td>integer</td>
<td>integer</td>
<td>integer</td>
</tr>
<tr>
<td>abs a</td>
<td>absolute value</td>
<td>integer</td>
<td>integer</td>
<td>integer</td>
</tr>
<tr>
<td>not a</td>
<td>negation</td>
<td>boolean, bit.</td>
<td>bit_vector</td>
<td>boolean, bit. bit_vector</td>
</tr>
<tr>
<td>a / b</td>
<td>division</td>
<td>integer</td>
<td>integer</td>
<td>integer</td>
</tr>
<tr>
<td>a mod b</td>
<td>modulo</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</td>
<td>identity</td>
<td>integer</td>
<td>integer</td>
<td>integer</td>
</tr>
<tr>
<td>- a</td>
<td>negation</td>
<td>integer</td>
<td>integer</td>
<td>integer</td>
</tr>
<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, 1-D array,</td>
<td>integer, 1-D array,</td>
<td>1-D array</td>
</tr>
<tr>
<td>a &amp; b</td>
<td>concatenation</td>
<td>1-D array, element</td>
<td>1-D array, element</td>
<td>1-D array</td>
</tr>
</tbody>
</table>
Operators in Standard VHDL

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>a sll b</td>
<td>shift left logical</td>
<td>bit.vector</td>
</tr>
<tr>
<td>a srl b</td>
<td>shift right logical</td>
<td>integer</td>
</tr>
<tr>
<td>a sla b</td>
<td>shift left arithmetic</td>
<td>bit.vector</td>
</tr>
<tr>
<td>a sra b</td>
<td>shift right arithmetic</td>
<td>integer</td>
</tr>
<tr>
<td>a rol b</td>
<td>rotate left</td>
<td>boolean</td>
</tr>
<tr>
<td>a ror b</td>
<td>rotate right</td>
<td>boolean</td>
</tr>
</tbody>
</table>

- **a == b**: equal to
- **a /= b**: not equal to
- **a < b**: less than
- **a <= b**: less than or equal to
- **a > b**: greater than
- **a >= b**: greater than or equal to

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>a and b</td>
<td>and</td>
<td>boolean.bit, bit.vector</td>
</tr>
<tr>
<td>a or b</td>
<td>or</td>
<td>same as a</td>
</tr>
<tr>
<td>a xor b</td>
<td>xor</td>
<td>boolean.bit, bit.vector</td>
</tr>
<tr>
<td>a nand b</td>
<td>nand</td>
<td>same as a</td>
</tr>
<tr>
<td>a nor b</td>
<td>nor</td>
<td>boolean.bit, bit.vector</td>
</tr>
<tr>
<td>a xnor b</td>
<td>xnor</td>
<td></td>
</tr>
</tbody>
</table>

IEEE std_logic_1164 Package

- What’s wrong with bit?
- New data type: std_logic, std_logic_vector
- std_logic:
  - 9 values: ('0', '1', 'Z', 'L', 'H', 'X', 'W', 'U', '-')
    - '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

- std_logic_vector
  - An array of elements with std_logic data type
  - Implies a bus
    ```vhd
    signal a: std_logic_vector(7 downto 0);
    ```
  - Another form (less desired)
    ```vhd
    signal a: std_logic_vector(0 to 7);
    ```
- Need to invoke package to use the data type
  ```vhd
  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>not a</td>
<td>std_logic_vector</td>
<td></td>
<td>same as a</td>
</tr>
<tr>
<td>a and 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_vector</td>
<td>std_logic</td>
<td>same as a</td>
</tr>
<tr>
<td>a xor b</td>
<td>std_logic_vector</td>
<td>same as a</td>
<td>same as a</td>
</tr>
<tr>
<td>a nand b</td>
<td>std_logic</td>
<td>std_logic</td>
<td></td>
</tr>
<tr>
<td>a nor b</td>
<td>std_logic_vector</td>
<td>same as a</td>
<td></td>
</tr>
<tr>
<td>a xnor b</td>
<td>std_logic</td>
<td>same as a</td>
<td></td>
</tr>
</tbody>
</table>
Type Conversion 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_stdulogic(a)</td>
<td>bit</td>
<td>std_logic</td>
</tr>
<tr>
<td>to_bit_vector(a)</td>
<td>std_logic_vector</td>
<td>bit_vector</td>
</tr>
<tr>
<td>to_stdlogicvector(a)</td>
<td>bit_vector</td>
<td>std_logic_vector</td>
</tr>
</tbody>
</table>

Type Conversion Example

```vhdl
signal s1, s2, s3: std_logic_vector(7 downto 0);
signal b1, b2: bit_vector(7 downto 0);
```

The following statements are wrong because of data type mismatch:

```vhdl
s1 <= b1;                 -- bit_vector assigned to std_logic_vector
b2 <= s1 and s2;          -- std_logic_vector assigned to bit_vector
s3 <= b1 or s2;           -- or is undefined between bit_vector
                        -- and std_logic_vector
```

We can use the conversion functions to correct these problems:

```vhdl
s1 <= to_stdlogicvector(b1);
b2 <= to_bitvector(s1 and s2);
s3 <= to_stdlogicvector(b1 or to_bitvector(s2));
```

The last statement can also be written as:

```vhdl
s3 <= to_stdlogicvector(b1 or to_bitvector(s2));
```
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, from the left most element.
  - All following returns true
    - "011"="011", "011">"010", "011">"00010", "0110">"011"

Concatenation Operator

- Concatenation operator (&)
  - y <= "00" & a(7 downto 2);
  - y <= a(7) & a(7) & a(7 downto 2);
  - y <= a(1 downto 0) & a(7 downto 2);
Array Aggregate

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

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
  
signal x, y: signed(15 downto 0);
- Need to invoke package to use the data type
  
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
### Overloaded Operators in IEEE numeric_std Package

<table>
<thead>
<tr>
<th>overloaded 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>abs a - a</td>
<td>absolute value negation</td>
<td>signed</td>
<td>signed</td>
<td>signed</td>
</tr>
<tr>
<td>a + b</td>
<td>arithmetic</td>
<td>unsigned</td>
<td>unsigned, natural</td>
<td>unsigned</td>
</tr>
<tr>
<td>a / b</td>
<td>arithmetic</td>
<td>unsigned, natural</td>
<td>unsigned</td>
<td>unsigned</td>
</tr>
<tr>
<td>a mod b</td>
<td>operation</td>
<td>signed</td>
<td>signed, integer</td>
<td>signed</td>
</tr>
<tr>
<td>a rem b</td>
<td>operation</td>
<td>signed, integer</td>
<td>signed</td>
<td>signed</td>
</tr>
<tr>
<td>a = b</td>
<td>relational</td>
<td>unsigned</td>
<td>unsigned, natural</td>
<td>boolean</td>
</tr>
<tr>
<td>a /= b</td>
<td>relational</td>
<td>unsigned, natural</td>
<td>unsigned</td>
<td>boolean</td>
</tr>
<tr>
<td>a &lt; b</td>
<td>operation</td>
<td>signed</td>
<td>signed, integer</td>
<td>boolean</td>
</tr>
<tr>
<td>a &lt;= b</td>
<td>operation</td>
<td>signed, integer</td>
<td>signed</td>
<td>boolean</td>
</tr>
<tr>
<td>a &gt; b</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a &gt;= b</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Overloaded Operators - Example

```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>unsigned</td>
<td>natural</td>
<td>same as a</td>
</tr>
<tr>
<td>shift_right(a, b)</td>
<td>shift right</td>
<td>signed</td>
<td>same as a</td>
<td></td>
</tr>
<tr>
<td>rotate_left(a, b)</td>
<td>rotate left</td>
<td>same as a</td>
<td>same as a</td>
<td></td>
</tr>
<tr>
<td>rotate_right(a, b)</td>
<td>rotate right</td>
<td>same as a</td>
<td>same as a</td>
<td></td>
</tr>
<tr>
<td>resize(a, b)</td>
<td>resize array</td>
<td>unsigned, signed</td>
<td>natural</td>
<td>same as a</td>
</tr>
<tr>
<td>std_match(a, b)</td>
<td>compare '=='</td>
<td>unsigned, signed</td>
<td>same as a</td>
<td>boolean</td>
</tr>
<tr>
<td>to_integer(a)</td>
<td>data type conversion</td>
<td>unsigned, signed</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>to_unsigned(a, b)</td>
<td>data type conversion</td>
<td>natural</td>
<td>natural</td>
<td>unsigned</td>
</tr>
<tr>
<td>to_signed(a, b)</td>
<td>data type conversion</td>
<td>integer</td>
<td>natural</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 are 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(a)</td>
</tr>
<tr>
<td>unsigned, std_logic_vector</td>
<td>unsigned</td>
<td>unsigned(a)</td>
</tr>
<tr>
<td>unsigned, signed</td>
<td>std_logic_vector</td>
<td>std_logic_vector(a)</td>
</tr>
<tr>
<td>unsigned, signed</td>
<td>integer</td>
<td>to_integer(a)</td>
</tr>
<tr>
<td>natural</td>
<td>unsigned</td>
<td>to_unsigned(a, size)</td>
</tr>
<tr>
<td>integer</td>
<td>signed</td>
<td>to_signed(a, size)</td>
</tr>
</tbody>
</table>

Type Conversion Example

```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);
```
Type Conversion Example

• Ok
  \[ u_3 \leq u_2 + u_1; \quad \text{--- ok, both operands unsigned} \]
  \[ u_4 \leq u_2 + 1; \quad \text{--- ok, operands unsigned and natural} \]

• Wrong
  \[ u_5 \leq s_g; \quad \text{-- type mismatch} \]
  \[ u_6 \leq 5; \quad \text{-- type mismatch} \]

• Fix
  \[ u_5 \leq \text{unsigned}(s_g); \quad \text{-- type casting} \]
  \[ u_6 \leq \text{to_unsigned}(5,4); \quad \text{-- conversion function} \]

Type Conversion Example

• Wrong
  \[ u_7 \leq s_g + u_1; \quad \text{-- + undefined over the types} \]

• Fix
  \[ u_7 \leq \text{unsigned}(s_g) + u_1; \quad \text{-- ok, but be careful} \]

• Wrong
  \[ s_3 \leq u_3; \quad \text{-- type mismatch} \]
  \[ s_4 \leq 5; \quad \text{-- type mismatch} \]

• Fix
  \[ s_3 \leq \text{std_logic_vector}(u_3); \quad \text{-- type casting} \]
  \[ s_4 \leq \text{std_logic_vector}(\text{to_unsigned}(5,4)); \]
Type Conversion Example

- Wrong
  \[
  s5 \leftarrow s2 + s1; \quad \text{-- undefined over std_logic_vector}
  
  s6 \leftarrow s2 + 1; \quad \text{-- undefined}
  \]
- Fix
  \[
  s5 \leftarrow \text{std_logic_vector(unsigned(s2) + unsigned(s1))};
  
  s6 \leftarrow \text{std_logic_vector(unsigned(s2) + 1)};
  \]

Non-IEEE Packages

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

- Software vendors frequently store them in ieee library
  
  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