VHDL Packages: numeric_std

- The numeric_std package
  - defines the unsigned and signed types based on the std_logic type
  - Defines numeric operations such as +, -, *, /, abs, >, <, etc. for these types
- Use the numeric_std package when need to perform arithmetic operations (or synthesize arithmetic operators) on std_logic types

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

Unsigned vs. Signed

- Unsigned is an unsigned binary integer with the the MSB as the left-most bit.
- signed is defined as a 2’s complement value with the most significant bit as the left-most bit.
  - This means the MSB of a unsigned(7 downto 0) is \( a(7) \)
  - The means the MSB of a unsigned(0 to 7) is \( a(0) \)
- Need signed, unsigned types because arithmetic results of operations can be different depending on the types.

Supported Operations

- Abs, unary -
- +, -, *, / (division), rem, mod
- \( \geq, \leq, >, <, >=, <= \)
- Shift_left, shift_right, rotate_left, rotate_right
- Sll, srl, rol, ror
- Resize
- To_integer, to_unsigned, to_signed
- Not, and, or, nand, nor, xor, xnor
- Std_match
- To_01

Conversions

Std_ulogic_vector, std_logic_vector, unsigned, signed are all closely related types (subtypes of std_ulogic).

Use explicit type casts when assigning one type to another

```vhdl
signal a_us, b_us: unsigned(7 downto 0);
signal a_s, b_s: signed(7 downto 0);
signal a, b: std_logic_vector(7 downto 0);
a <= std_logic_vector(a_s);
a_s <= signed(a_us);
a_us <= unsigned(a);
b_s <= signed(a);
```

Type cast specifies name of target type.

Metalogical and ‘Z’ Values

- A metalogical value is defined as ‘X’, ‘W’, ‘U’, or ‘-’
- A high impedance value is ‘Z’
- If any bit in an operand to a numeric_std function contains a metalogical or high impedance value (‘Z’), the result is returned with all bits set to ‘X’.
  - One exception, the ‘std_match’ function
- A value is well-defined if it contains no metalogical or high impedance values.

Integer Conversions

```vhdl
function TO_INTEGER (ARG: UNSIGNED) return NATURAL;
function TO_INTEGER (ARG: SIGNED) return INTEGER;
function TO_UNSIGNED (ARG, SIZE: NATURAL) return UNSIGNED;
function TO_SIGNED (ARG: INTEGER; SIZE: NATURAL) return SIGNED;
```

Basically the same functions as in the std_logic_1164 package.
Different forms of ‘+’

function ‘+’ (L, R: UNSIGNED) return UNSIGNED;
function ‘+’ (L, R: SIGNED) return SIGNED;
function ‘+’ (L: UNSIGNED; R: NATURAL) return UNSIGNED;
function ‘+’ (L: NATURAL; R: UNSIGNED) return UNSIGNED;
function ‘+’ (L: INTEGER; R: SIGNED) return SIGNED;
function ‘+’ (L: SIGNED; R: INTEGER) return SIGNED;

Note different combinations of allowable operands.

For synthesis, there is a problem – do not have access to carry-in, or carry-out which would be very useful. Would have to use operands with 2-extra bits to get access to both carry-in and carry-out.

Mixed Signed/Unsigned Operands

• Note that the defined forms of ‘+’ do not have mixed unsigned/signed operands
• Must do explicit conversions if want to do mix unsigned/signed operands
• This way the user decides how the sign bit should be handled.

Mixed Signed/Unsigned Operands

• Note that the defined forms of ‘+’ do not have mixed unsigned/signed operands
• Must do explicit conversions if want to do mix unsigned/signed operands
• This way the user decides how the sign bit should be handled.

Mixed Signed/Unsigned Operands

• Note that the defined forms of ‘+’ do not have mixed unsigned/signed operands
• Must do explicit conversions if want to do mix unsigned/signed operands
• This way the user decides how the sign bit should be handled.

Mixed Signed/Unsigned Operands

• Note that the defined forms of ‘+’ do not have mixed unsigned/signed operands
• Must do explicit conversions if want to do mix unsigned/signed operands
• This way the user decides how the sign bit should be handled.

Mixed Signed/Unsigned Operands

• Note that the defined forms of ‘+’ do not have mixed unsigned/signed operands
• Must do explicit conversions if want to do mix unsigned/signed operands
• This way the user decides how the sign bit should be handled.

Mixed Signed/Unsigned Operands

• Note that the defined forms of ‘+’ do not have mixed unsigned/signed operands
• Must do explicit conversions if want to do mix unsigned/signed operands
• This way the user decides how the sign bit should be handled.

Mixed Signed/Unsigned Operands

• Note that the defined forms of ‘+’ do not have mixed unsigned/signed operands
• Must do explicit conversions if want to do mix unsigned/signed operands
• This way the user decides how the sign bit should be handled.

Mixed Signed/Unsigned Operands

• Note that the defined forms of ‘+’ do not have mixed unsigned/signed operands
• Must do explicit conversions if want to do mix unsigned/signed operands
• This way the user decides how the sign bit should be handled.

Mixed Signed/Unsigned Operands

• Note that the defined forms of ‘+’ do not have mixed unsigned/signed operands
• Must do explicit conversions if want to do mix unsigned/signed operands
• This way the user decides how the sign bit should be handled.

Mixed Signed/Unsigned Operands

• Note that the defined forms of ‘+’ do not have mixed unsigned/signed operands
• Must do explicit conversions if want to do mix unsigned/signed operands
• This way the user decides how the sign bit should be handled.

Mixed Signed/Unsigned Operands

• Note that the defined forms of ‘+’ do not have mixed unsigned/signed operands
• Must do explicit conversions if want to do mix unsigned/signed operands
• This way the user decides how the sign bit should be handled.

Mixed Signed/Unsigned Operands

• Note that the defined forms of ‘+’ do not have mixed unsigned/signed operands
• Must do explicit conversions if want to do mix unsigned/signed operands
• This way the user decides how the sign bit should be handled.

Mixed Signed/Unsigned Operands

• Note that the defined forms of ‘+’ do not have mixed unsigned/signed operands
• Must do explicit conversions if want to do mix unsigned/signed operands
• This way the user decides how the sign bit should be handled.

Mixed Signed/Unsigned Operands

• Note that the defined forms of ‘+’ do not have mixed unsigned/signed operands
• Must do explicit conversions if want to do mix unsigned/signed operands
• This way the user decides how the sign bit should be handled.

Mixed Signed/Unsigned Operands

• Note that the defined forms of ‘+’ do not have mixed unsigned/signed operands
• Must do explicit conversions if want to do mix unsigned/signed operands
• This way the user decides how the sign bit should be handled.

Mixed Signed/Unsigned Operands

• Note that the defined forms of ‘+’ do not have mixed unsigned/signed operands
• Must do explicit conversions if want to do mix unsigned/signed operands
• This way the user decides how the sign bit should be handled.

Mixed Signed/Unsigned Operands

• Note that the defined forms of ‘+’ do not have mixed unsigned/signed operands
• Must do explicit conversions if want to do mix unsigned/signed operands
• This way the user decides how the sign bit should be handled.

Mixed Signed/Unsigned Operands

• Note that the defined forms of ‘+’ do not have mixed unsigned/signed operands
• Must do explicit conversions if want to do mix unsigned/signed operands
• This way the user decides how the sign bit should be handled.

Mixed Signed/Unsigned Operands

• Note that the defined forms of ‘+’ do not have mixed unsigned/signed operands
• Must do explicit conversions if want to do mix unsigned/signed operands
• This way the user decides how the sign bit should be handled.

Mixed Signed/Unsigned Operands

• Note that the defined forms of ‘+’ do not have mixed unsigned/signed operands
• Must do explicit conversions if want to do mix unsigned/signed operands
• This way the user decides how the sign bit should be handled.

Mixed Signed/Unsigned Operands

• Note that the defined forms of ‘+’ do not have mixed unsigned/signed operands
• Must do explicit conversions if want to do mix unsigned/signed operands
• This way the user decides how the sign bit should be handled.

Mixed Signed/Unsigned Operands

• Note that the defined forms of ‘+’ do not have mixed unsigned/signed operands
• Must do explicit conversions if want to do mix unsigned/signed operands
• This way the user decides how the sign bit should be handled.

Mixed Signed/Unsigned Operands

• Note that the defined forms of ‘+’ do not have mixed unsigned/signed operands
• Must do explicit conversions if want to do mix unsigned/signed operands
• This way the user decides how the sign bit should be handled.

Mixed Signed/Unsigned Operands

• Note that the defined forms of ‘+’ do not have mixed unsigned/signed operands
• Must do explicit conversions if want to do mix unsigned/signed operands
• This way the user decides how the sign bit should be handled.

Mixed Signed/Unsigned Operands

• Note that the defined forms of ‘+’ do not have mixed unsigned/signed operands
• Must do explicit conversions if want to do mix unsigned/signed operands
• This way the user decides how the sign bit should be handled.
VHDL Synthesizeable Subset

- The subset of VHDL that is synthesizable is limited
  - Unfortunately, not all synthesis tools agree on what statements are 'synthesizeable'
- IEEE std 1076.6-1999 defines the RTL synthesizable subset of the language and how language statements should be interpreted for synthesis purposes.
  - This is important as it enhances the portability of RTL models across synthesis tools from different vendors.
  - Much intellectual property of standard bus interfaces and building blocks (e.g., PCI, Firewire, USB, CPUs) are distributed in RTL form so want the RTL to be as portable as possible.

An Example of Coding for Simulation vs. Synthesis

A:  ```vhdl```
```vhdl```
```
process (clk)
if (clk'event and clk = '1') then
  q <= d;
end if;
```
```vhdl```

B:  ```vhdl```
```vhdl```
```
process (clk)
if (clk = '1') then
  q <= d;
end if;
```
```vhdl```

Processes A and B both simulate exactly the same.

For simulation, do not need clk'event because we know that a clk'event triggers the process.

The synthesis tool needs clk'event to infer that an edge-triggered device should be synthesized, not a latch!!

pragmas

- A pragma is an element of a language that is intended be interpreted as a command that controls the action of the compiler or synthesis tools
- The RTL synthesis standard defines two types of pragmas
  - Attributes
  - Metacomments
- A metacomment is a pragma embedded in a comment statement. Two meta comments are defined
  -- RTL_SYNTHESIS OFF
  -- RTL_SYNTHESIS ON
This causes statements bracketed by these metacomments to be ignored by the synthesis tool.

(pragmas cont.)

- Only one attribute is defined as having a synthesis specific interpretation
- The ENUM_ENCODING attribute is used specify the encoding for a enumerated type
  - One use is to specify encoding for FSM states
```vhdl```
```
type mystate is (ST0,ST1,ST2,ST3);
attribute ENUM_ENCODING of mystate:
  type is "0001 0010 0100 1000";
```
```vhdl```
\[\text{Defines encoding for ST0 as "001", ST1 as "0010", etc.}\]

RTL Synthesizeable standard

- The standard is basically the LRM with statements marked out as being not recognized by a synthesis tool
- Most things are obvious
  - i.e., file operations, dynamic memory allocation are not synthesizable
  - Floating point types are not synthesizable
  - Delays on signals are ignored
- Arrays and Record types are fine as long as each field specifies a synthesizable type
- Look at the standard for more detail