Fixed and Floating Point Packages

By

Jim Lewis, SynthWorks VHDL Training, jim@synthworks.com

David Bishop, Kodak, dbishop@vhdl.org
Fixed and Floating Point Packages

- Fixed Point
  - Package & Types
  - Format
  - Sizing & Overloading
  - Literals in Assignments and Expressions
  - Quirks

- Floating Point
  - Package & Types
  - Format
  - Sizing & Overloading
  - Literals in Assignments and Expressions

Caution:
These packages are a work in progress. They are based on numeric_std, so support is good but not perfect.
Fixed Point Types

Library ieee_proposed;                 -- work in progress
use ieee_proposed.fixed_pkg.all;

- ufixed = unsigned fixed point
  type ufixed is array (integer range <>) of std_logic;

- sfixed = signed fixed point
  type sfixed is array (integer range <>) of std_logic;
**Fixed Point Format**

- Range is required to be downto
- Whole number is on the left and includes 0 index (3 downto 0)
- Fraction is to the right of the 0 index (-1 downto -3)
- Ok to be only an integer or only a fraction

```
constant A : ufixed(3 downto -3) := "0110100";

3210 -3
IIIIFFF
0110100 = 0110.100 = 6.5
```
Fixed Point is Full Precision Math

```vhdl
signal A4_3, B4_3 : ufixed ( 3 downto -3 ) ;
signal Y5_3       : ufixed ( 4 downto -3 ) ;

Y5_3 <= A4_3 + B4_3 ;
```

- Details on results sizes are in the fixed point users guide
- Note that in numeric_std addition/subtraction is modulo math

<table>
<thead>
<tr>
<th>Arithmetic Overloading</th>
<th>Ufixed Result</th>
<th>Sfixed Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ - * / rem mod abs</td>
<td>ufixed op ufixed</td>
<td>sfixed op sfixed</td>
</tr>
<tr>
<td>= /= &gt; &lt; &gt;= &lt;=</td>
<td>ufixed op real</td>
<td>sfixed op real</td>
</tr>
<tr>
<td></td>
<td>real op ufixed</td>
<td>real op sfixed</td>
</tr>
<tr>
<td></td>
<td>ufixed op natural</td>
<td>sfixed op integer</td>
</tr>
<tr>
<td></td>
<td>natural op ufixed</td>
<td>integer op sfixed</td>
</tr>
</tbody>
</table>

Lewis
## Literals in Assignments

```vhdl
signal A4 : ufixed (3 downto -3) ;

-- String Literal
A4 <= "0110100" ; -- 6.5

-- Real and/or Integer Literal
A4 <= to_ufixed( 6.5, A4 ) ; -- sized by A4
A4 <= to_ufixed( 6.5, 3, -3 ) ; -- pass indicies
```

- **To_ufixed**
  - Size of result based on range of an argument (such as A4) or by passing the indices (3, -3)
  - Overloaded to accept either real or integer numbers
  - Type real and integer limited the precision of the literal
**Issue:** A string literal used in an expression has range based on the direction of the base type and left index (integer'low)

```vhdl
signal A4    : ufixed (3 downto -3) ;
signal Y5    : ufixed (4 downto -3) ;

-- Y5 <= A4 + "0110100" ; -- illegal,
-- ^^indices are integer'low to ...
```

**Solutions**

```vhdl
subtype ufixed4_3 is ufixed (3 downto -3) ;
signal A4, B4 : ufixed4_3 ;
signal Y5     : ufixed (4 downto -3) ;

Y5 <= A4  +  ufixed4_3'("0110100") ;  
Y5 <= A4  +  6.5 ; -- overloading
Y5 <= A4  +  6 ;
```
Quirks: Accumulator

- Size of result needs to match size of one of inputs

```vhdl
signal A4 : ufixed (3 downto -3); 
signal Y7 : ufixed (6 downto -3); 

-- Y7 <= Y7 + A; -- illegal, result too big

-- Solution, resize the result
Y7 <= resize ( 
    arg => Y7 + A, 
    size_res => Y7, 
    overflow_style => fixed_saturate,  
                      -- fixed_wrap 
    round_style => fixed_round 
                        -- fixed_truncate
);
```
## Fixed Point Conversions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>To_ufixed</td>
<td>integer, real, unsigned, std_logic_vector to ufixed</td>
</tr>
<tr>
<td>To_sfixed</td>
<td>integer, real, signed, std_logic_vector to sfixed</td>
</tr>
<tr>
<td>Resize</td>
<td>ufixed to ufixed or sfixed to sfixed both with potential rounding</td>
</tr>
<tr>
<td>Add_sign</td>
<td>ufixed to sfixed</td>
</tr>
<tr>
<td>to_real</td>
<td>ufixed or sfixed to real (scalar)</td>
</tr>
<tr>
<td>to_integer</td>
<td>ufixed or sfixed to integer (scalar)</td>
</tr>
<tr>
<td>to_unsigned</td>
<td>ufixed to unsigned (array)</td>
</tr>
<tr>
<td>to_signed</td>
<td>sfixed to signed (array)</td>
</tr>
<tr>
<td>to_slv</td>
<td>ufixed or sfixed to slv (array) for top level ports</td>
</tr>
</tbody>
</table>
Floating Point Types

Library ieee_proposed ; -- work in progress
use ieee_proposed.float_pkg.all ;

- Main type is unconstrained:
  
  type float is array (integer range <>) of std_logic;

- Package also defines subtypes:
  
  - IEEE 754 Single Precision
    
    subtype float32 is float( 8 downto -32);

  - IEEE 754 Double Precision
    
    subtype float64 is float(11 downto -52);

  - IEEE 854 Extended Precision
    
    subtype float128 is float(15 downto -112);
Floating Point Format

```vhdl
signal A, B, Y : float (8 downto -23) ;
```

| 8  | 76543210 | 12345678901234567890123 |
| S  | EEEEEEEE | FFFFFFFFFFFFFFFFFFFFFFF |

E = Exponent is biased by 127
F = Fraction has an implied 1 in leftmost bit

value = $2^{(E-127)} \times (1 + F)$

0 10000001 10100000000000000000000000000000 = +1 * $2^{(129 - 127)} \times (1.0 + 0.5 + 0.125)$

= +1 * $2^{2}$ * (1.625) = 6.5

- Range is required to be downto
- Sign Bit = A'left = bit 8 (0 = positive, 1 = negative)
- Exponent = range A'left - 1 downto 0 = 7 downto 0
- Mantissa = range -1 downto A'right = -1 downto -23
- Sign, Exponent and Mantissa are always present
Range of Values

- Large positive number (Exponent of all 1 is reserved)

  \[
  0\ 11111110\ 000000000000000000000000
  = +1 \times 2^{(254 - 127)} \times (1.0 + 0)
  = 2^{(127)}
  \]

- Smallest positive number without denormals

  \[
  0\ 00000001\ 00000000000000000000000000000000
  = +1 \times 2^{(1 - 127)} \times (1.0 + 0)
  = 2^{(-126)}
  \]

- Extended small numbers = Denormals, but only when enabled

  \[
  0\ 00000000\ 10000000000000000000000000000000
  = +1 \times 2^{(1 - 127)} \times (0 + 0.5)
  = +1 \times 2^{(-126)} \times 2^{(-1)}
  = 2^{(-127)}
  \]
Special Numbers

- Zero (Positive 0 = Negative 0)

<table>
<thead>
<tr>
<th>Positive</th>
<th>Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 00000000000000000000000000000000000000000</td>
<td>1 00000000000000000000000000000000000000000</td>
</tr>
</tbody>
</table>

- Infinity

<table>
<thead>
<tr>
<th>Positive</th>
<th>Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1111111111111111111111111111111111111111111</td>
<td>1 111111111111111111111111111111111111111111</td>
</tr>
</tbody>
</table>

- NAN - Not A Number

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 111111111111111111111111111111111111111111</td>
</tr>
</tbody>
</table>

- Exponent with all 0 is reserved for zero and denormal numbers
- Exponent with all 1 is reserved for infinity and NAN
Floating Point Types

signal A32, B32, Y32 : float (8 downto -23) ;

. . .

Y32 <= A32 + B32 ;

- Floating point result will have the maximum exponent and maximum mantissa of its input arguments.

- Also need to specify:
  - Rounding Default = round_nearest
    - Round nearest, Round 0, Round +inf, round -inf
  - Denormals: On / Off Default = on = true
  - Check NAN and Overflow Default = on = true
  - Guard Bits: Extra bits for rounding. Default = 3

- Current package uses constants - rather limiting
- Long term plan is to use generics - a new feature being added in 2006
## Overloading

<table>
<thead>
<tr>
<th>Arithmetic Overloading</th>
<th>Float Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ - * / rem mod abs</td>
<td>float op float</td>
</tr>
<tr>
<td>= /= &gt; &lt; &gt;= &lt;=</td>
<td>float op real</td>
</tr>
<tr>
<td></td>
<td>real op float</td>
</tr>
<tr>
<td></td>
<td>float op integer</td>
</tr>
<tr>
<td></td>
<td>integer op float</td>
</tr>
</tbody>
</table>
**Literals in Assignments**

```vhdl
signal A_fp32 : float32;
.
-- String Literal
A_fp32 <= "01000000110100000000000000000000" ; -- 6.5

-- Real and/or Integer Literal
A_fp32 <= to_float(6.5, A_fp32); -- size using A_fp32
A_fp32 <= to_float(6.5, 8, -32); -- pass indicies
```

- **To_float**
  - Needs to size the result based on range of an argument (such as A_fp32) or by passing the indicies (8, -32)
  - Overloaded to accept either integers or real numbers
  - Note the required precision of type real and integer is limited by the language
Literals in Expressions

- **Issue:** A string literal used in an expression has range based on the direction of the base type and left index (integer'low)

```vhdl
signal A, Y : float32;

-- Y <= A + "01000001101000000000000000000000"; -- ill
-- ^^ range integer'low to ...
```

- **Solutions**

```vhdl
signal A, Y : float32;

Y <= A + float32'("01000001101000000000000000000000");
Y <= A + 6.5;   -- overloading
Y <= A + 6;     -- overloading
```
Floating Point Conversions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>To_float</td>
<td>integer, real, ufixed, sfixed, signed, unsigned, and std_logic_vector to float</td>
</tr>
<tr>
<td>Resize</td>
<td>float to float with potential rounding, ...</td>
</tr>
<tr>
<td>to_real</td>
<td>float to real (scalar)</td>
</tr>
<tr>
<td>to_integer</td>
<td>float to integer (scalar)</td>
</tr>
<tr>
<td>to_sfixed</td>
<td>float to sfixed (array)</td>
</tr>
<tr>
<td>to_ufixed</td>
<td>float to ufixed (array)</td>
</tr>
<tr>
<td>to_unsigned</td>
<td>float to unsigned (array)</td>
</tr>
<tr>
<td>to_signed</td>
<td>float to signed (array)</td>
</tr>
<tr>
<td>to_slv</td>
<td>float to slv (array) for top level ports</td>
</tr>
</tbody>
</table>
Going Further

- Package standardization not complete.
- Ok to use since it is based on numeric_std

- Current packages and documentation available at:
  http://www.eda.org/vhdl-200x/vhdl-200x-ft/packages/files.html

- Current methodology
  - Create a library named: ieee_proposed
  - Copy fixed_pkg and float_pkg to another name.
  - Set the constants to appropriate values
  - Compile into the library
  - For different constant settings, create additional copies of the packages with different names

- With package generics (planned for VHDL-2006),
  - package instantiations replace copies of a package with constants