

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

SynthWorks



- 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



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

  3210 -3
  IIIIFFF
  0110100 = 0110.100 = 6.5
```

- 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 a integer or only a fraction

Fixed Point is Full Precision Math



```
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

| Arithmetic Overloading | Ufixed Result | Sfixed Result |
|---------------------------------------|--|--|
| + - * / rem mod abs = /= > < >= <= | ufixed op ufixed ufixed op real real op ufixed ufixed op natural natural op ufixed | sfixed op sfixed sfixed op real real op sfixed sfixed op integer integer op sfixed |

Literals in Assignments



```
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 indicies (3, -3)
 - Overloaded to accept either real or integer numbers
 - Type real and integer limited the precision of the literal



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)

```

signal A4      : ufixed (3 downto -3) ;
signal Y5      : ufixed (4 downto -3) ;
. . .
-- Y5 <= A4 + "0110100" ;    -- illegal,
--                ^^indices are integer'low to ...

```

- Solutions

```

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

```

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
);

```



| | |
|--------------------|--|
| To_ufixed | integer, real, unsigned, std_logic_vector to ufixed |
| To_sfixed | integer, real, signed, std_logic_vector to sfixed |
| Resize | ufixed to ufixed or sfixed to sfixed both with potential rounding |
| Add_sign | ufixed to sfixed |
| to_real | ufixed or sfixed to real (scalar) |
| to_integer | ufixed or sfixed to integer (scalar) |
| to_unsigned | ufixed to unsigned (array) |
| to_signed | sfixed to signed (array) |
| to_slv | ufixed or sfixed to slv (array) for top level ports |

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 -23);
```

- 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



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

```

8  76543210  12345678901234567890123
S  EEEEEEEEE FFFFFFFFFFFFFFFFFFFFFFFF

```

E = Exponent is biased by 127

F = Fraction has an implied 1 in leftmost bit

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

```

0  10000001  101000000000000000000000
= +1 * 2**(129 - 127) * (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 * 2**(254 - 127) * (1.0 + 0)
= 2**(127)

```

- Smallest positive number without denormals

```

0  00000001  000000000000000000000000
= +1 * 2**(1 - 127) * (1.0 + 0)
= 2**(-126)

```

- Extended small numbers = Denormals, but only when enabled

```

0  00000000  100000000000000000000000
= +1 * 2**(1 - 127) * (0 + 0.5)
= +1 * 2**(-126) * 2**(-1)
= 2 ** (-127)

```



Special Numbers

- Zero (Positive 0 = Negative 0)

```
0 00000000 000000000000000000000000 -- Positive
1 00000000 000000000000000000000000 -- Negative
```

- Infinity

```
0 11111111 000000000000000000000000 -- Positive
1 11111111 000000000000000000000000 -- Negative
```

- NAN - Not A Number

```
1 11111111 000000000000000000000001
```

- 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



| Arithmetic Overloading | Float Result |
|---------------------------------------|--|
| + - * / rem mod abs = /= > < >= <= | float op float float op real real op float float op integer integer op float |

Literals in Assignments



```
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)

```

signal A, Y : float32 ;
. . .
-- Y <= A + "01000000110100000000000000000000"; -- ill
--          ^^ range integer'low to ...

```

- Solutions

```

signal A, Y : float32 ;
. . .
Y <= A + float32'("01000000110100000000000000000000");
Y <= A + 6.5 ;      -- overloading
Y <= A + 6 ;       -- overloading

```

Floating Point Conversions



| | |
|--------------------|---|
| To_float | integer, real, ufixed, sfixed, signed, unsigned, and std_logic_vector to float |
| Resize | float to float with potential rounding, ... |
| to_real | float to real (scalar) |
| to_integer | float to integer (scalar) |
| to_sfixed | float to sfixed (array) |
| to_ufixed | float to ufixed (array) |
| to_unsigned | float to unsigned (array) |
| to_signed | float to signed (array) |
| to_slv | float to slv (array) for top level ports |



- 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

About SynthWorks VHDL Training

Comprehensive VHDL Introduction 4 Days

http://www.synthworks.com/comprehensive_vhdl_introduction.htm

A design and verification engineer's introduction to VHDL syntax, RTL coding, and testbenches. Students get VHDL hardware experience with our FPGA based lab board.

VHDL Testbenches and Verification 3 days

http://www.synthworks.com/vhdl_testbench_verification.htm

Learn to simplify writing tests by creating transaction-based testbenches.

Intermediate VHDL Coding for Synthesis 2 Days

http://www.synthworks.com/intermediate_vhdl_synthesis.htm

Learn RTL (hardware) coding styles that produce better, faster, and smaller logic.

Advanced VHDL Coding for Synthesis 2 Days

http://www.synthworks.com/advanced_vhdl_synthesis.htm

Learn to avoid RTL coding issues, problem solving techniques, and advanced VHDL constructs.

For additional courses see: <http://www.synthworks.com>